IMPLEMENTING AND TRAINING NEURAL NETWORKS(CONT.)

STFANO MELACCI

UNVERSITY OF SIENA

DERIVATIVES: SCALARS, VECTORS, MATRICES, TENSORS

- Let's consider a scalar valued function (we will always assume values to be in the set of real numbers)
  - f(x), where x is a scalar
  - f(**x**), where **x** is a vector (*n* elements)
  - f(X), where X is a matrix (*n* x *m*)
  - f(X), where X is a "tensor" (*n* x *m* x *c* – *or any other*)

$$\mathbf{x} = [x_1, \ldots, x_n]^T$$

$$X = \begin{bmatrix} X_{11}, & \ldots, & X_{1n} \\ \ldots, & \ldots, & \ldots \\ X_{m1}, & \ldots, & X_{mn} \end{bmatrix}$$

$$X = \begin{bmatrix} [X_{111}, \ldots, X_{11c}] & \ldots, & [X_{1n1}, \ldots, X_{1nc}] \\ \ldots, & \ldots, & \ldots \\ [X_{m11}, \ldots, X_{m1c}], & \ldots, & [X_{mn1}, \ldots, X_{mnc}] \end{bmatrix}$$

Assumption: if we imagine to assing an unique numerical ID to each element of X, then the last dimension of X is the one on which elements are "contiguous" – explain this!

DERIVATIVES: SCALARS, VECTORS, MATRICES, TENSORS

- Let's consider a scalar valued function (we will always assume values to be in the set of real numbers)
  - f(x), where x is a scalar
  - f(**x**), where **x** is a vector (*n* elements)
  - f(X), where X is a matrix (*m* x *n*)
  - f(X), where X is a "tensor" (*m* x *n* x *c*)

most common and straightforward case

$$\frac{df(x)}{dx} \qquad \frac{df(\mathbf{x})}{d\mathbf{x}} \qquad \frac{df(X)}{dX}$$

DERIVATIVES W.R.T. VECTOR/MATIX

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \quad \cdots, \quad \frac{\partial f(\mathbf{x})}{\partial x_n} \right]^T$$

Denominator Layout

$$\frac{df(X)}{dX} = \begin{bmatrix} \frac{\partial f(X)}{\partial X_{11}}, & \cdots, & \frac{\partial f(X)}{\partial X_{1n}} \\ \frac{\partial f(X)}{\partial X_{21}}, & \cdots, & \frac{\partial f(X)}{\partial X_{2n}} \\ \cdots, & \cdots, & \cdots \\ \frac{\partial f(X)}{\partial X_{m1}}, & \cdots, & \frac{\partial f(X)}{\partial X_{mn}} \end{bmatrix}$$

DERIVATIVES W.R.T. TENSOR: GENERAL CASE

- The derivative of a scalar function w.r.t. to any kind of input tensorial structure X is a tensor with the same dimensions of X, where each element is the partial derivative of the function w.r.t. to the corresponding element of X
  - WARNING: I will frequently relax the formality of several descriptions (and notation) with the purpose of making them more accessible

Denominator Layout

$$\frac{df(X)}{dX} = \left[\frac{\partial f(X)}{\partial X_{ijk}}, \quad \begin{matrix} i = 1, \ldots, m \\ j = 1, \ldots, n \\ k = 1, \ldots, c \end{matrix} \right]$$

It generalizes all the cases considered so far!
(it holds also for 4D, 5D, whateverD tensors)

DERIVATIVES: LET'S RAISE THE BAR

- Let's consider a **vector-valued function** that outputs a vector with *a* elements
  - **f**(X)
- What it we also consider a function that outputs an *a x b* matrix?
  - F(X)
- What is we consider a function that returns a 3D structure of size *a* x *b* x *u*?
  - F(X)
  - *most generic case*

DERIVATIVES: LET'S RAISE THE BAR

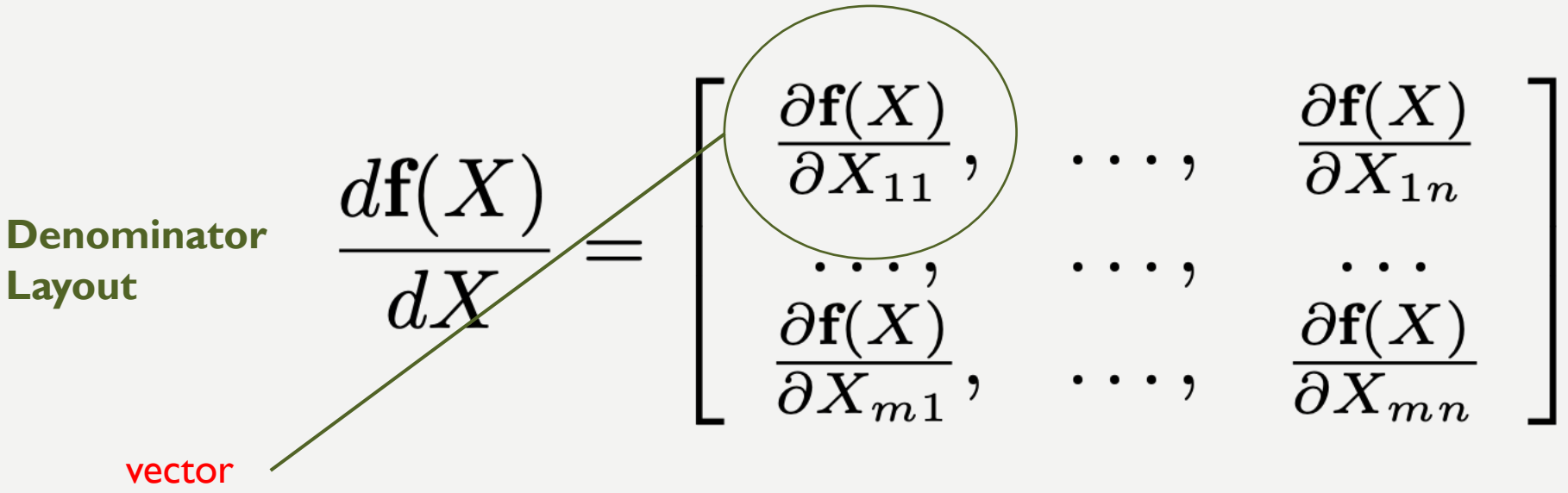
$$\frac{d\mathbf{f}(x)}{dx} = \left[\frac{\partial f_1(x)}{\partial x}, \quad \cdots, \quad \frac{\partial f_a(x)}{\partial x} \right]$$

Denominator Layout

$$\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1}, & \cdots, & \frac{\partial f_a(\mathbf{x})}{\partial x_1} \\ \cdots, & \cdots, & \cdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_n}, & \cdots, & \frac{\partial f_a(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Jacobian Matrix

DERIVATIVES: LET'S RAISE THE BAR



DERIVATIVES: LET'S RAISE THE BAR

- The derivative of a vectorial function w.r.t. to any kind of input tensorial structure X is a tensor whose "first" dimensions are the same dimensions of X
  - They index partial derivatives of the function w.r.t. to the corresponding elements of X
- The "last" dimensions are the ones of F(X)
  - They are used to index the output elements of F(X)

$$
\begin{array}{lcl}
X & : & m \times n \times c \\
F(X) & : & a \times b \times u \\
\frac{dF(X)}{dX} & : & \underbrace{m \times n \times c}_{\text{shape of X}} \times \underbrace{a \times b \times u}_{\text{shape of F(X)}}
\end{array}
$$

Denominator Layout

$$
\frac{\partial f(X)}{\partial X} = \left[\frac{\partial f_{stz}(X)}{\partial X_{ijk}}, \begin{array}{l} i = 1, \ldots, m \\ j = 1, \ldots, n \\ k = 1, \ldots, c \\ s = 1, \ldots, a \\ t = 1, \ldots, b \\ z = 1, \ldots, u \end{array} \right]
$$

CHAIN RULE

- Before diving into further tensor-related stuff, let's go back to a simple concept
- Whenever we are given a composite function *f(g(x))*, then the derivative of *f* with respect to *x* can be written as follows

$$z = g(x)$$

$$y = f(g(x)) = f(z)$$

$$\frac{df(g(x))}{dx} = \frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$

The whole derivative has been decomposed as the product of the derivatives of two "computational blocks/nodes"

CHAIN RULE

- Before diving into further tensor-related stuff, let's go back to a simple concept
- Whenever we are given a composite function *f(g(x))*, then the derivative of *f* with respect to *x* can be written as follows

$$z = g(x)$$

$$y = f(g(x)) = f(z)$$

$$\frac{df(g(x))}{dx} = \frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$

Computational block/node 1:
y = f(z)

(we differentiate the block with respect to its input *z*)

CHAIN RULE

- Before diving into further tensor-related stuff, let's go back to a simple concept
- Whenever we are given a composite function *f(g(x))*, then the derivative of *f* with respect to *x* can be written as follows

$$z = g(x)$$

$$y = f(g(x)) = f(z)$$

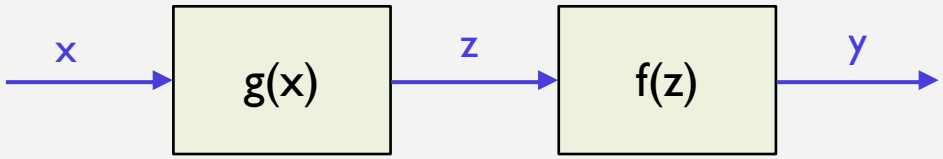
$$\frac{df(g(x))}{dx} = \frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$

Computational block/node 2:
z = g(x)

(we differentiate the block with respect to its input *x*)

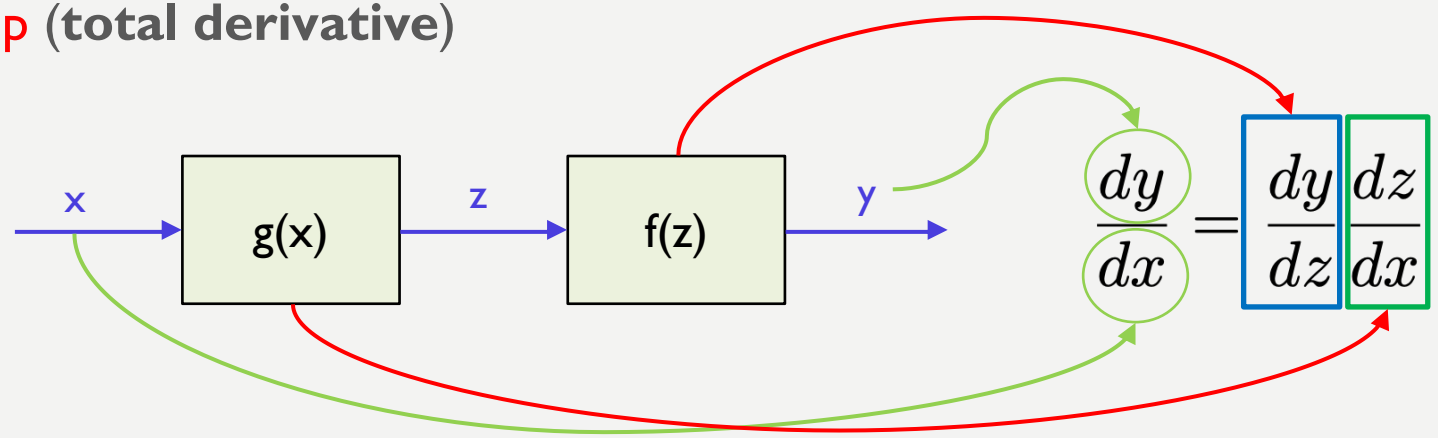
COMPUTATIONAL GRAPHS

- A computational graph describes a mathematical expression using a graph-based representation (directed graph) in which **nodes** are *operations* and **edges** are *data*
  - A node could be about a very low-level operation (a sum) or about a more "complicated" function - it is a design choice
  - In the case of the previous slides, $f(g(x))$, we get:



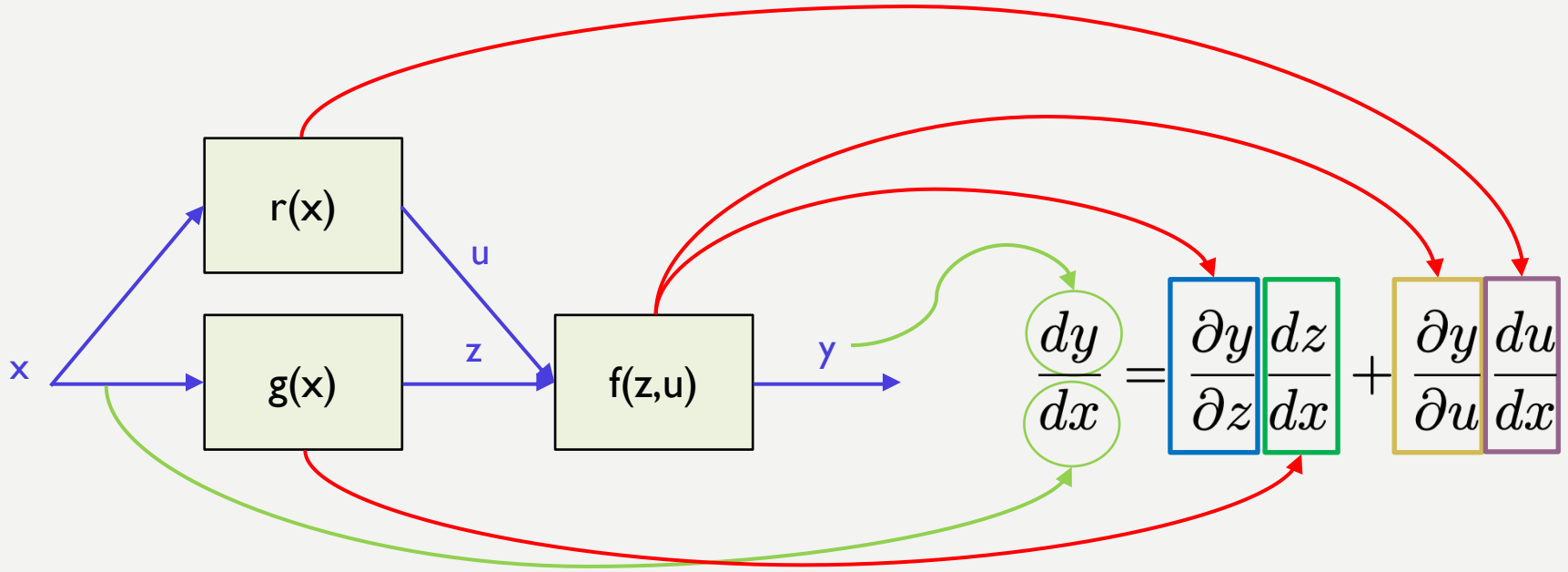
COMPUTATIONAL GRAPHS

- We can compute the derivative of some data **a** (edge) with respect to some other data **b** (edge) using the *chain rule*, following the path that connects **a** to **b**, and making products of the derivatives of each node we encounter, with respect to the data in the path
  - If there is not path, then the derivative is trivially zero
  - If there are multiple paths going from **a** to **b**, then the derivatives computed over all the paths must be summed up (**total derivative**)



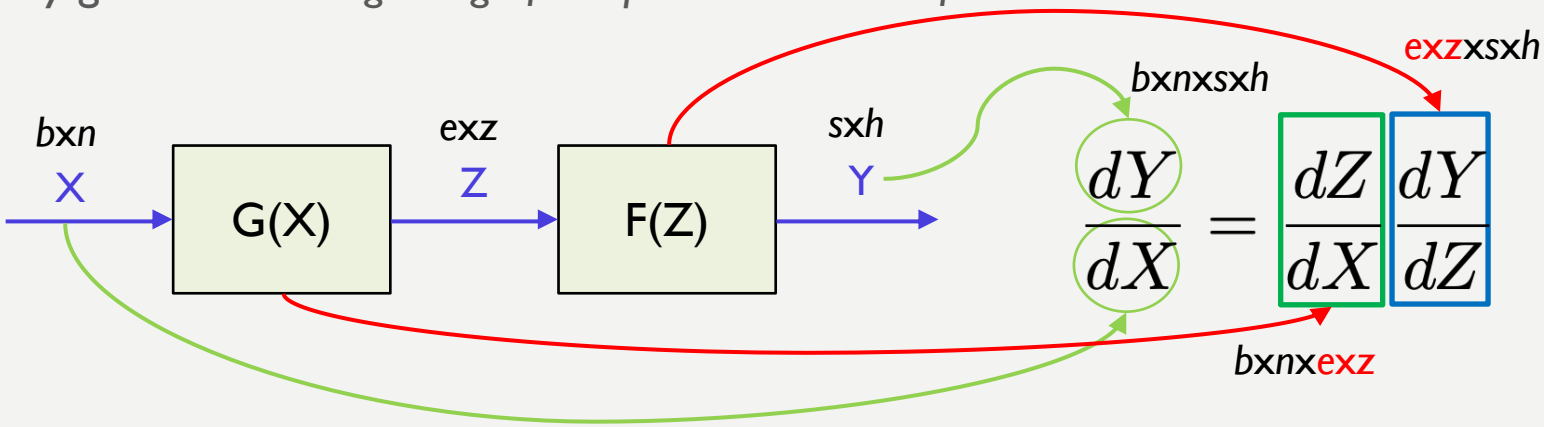
COMPUTATIONAL GRAPHS

- Another example (recall: If there are multiple paths going from **a** to **b**, then the derivatives computed over all the paths must be summed up)



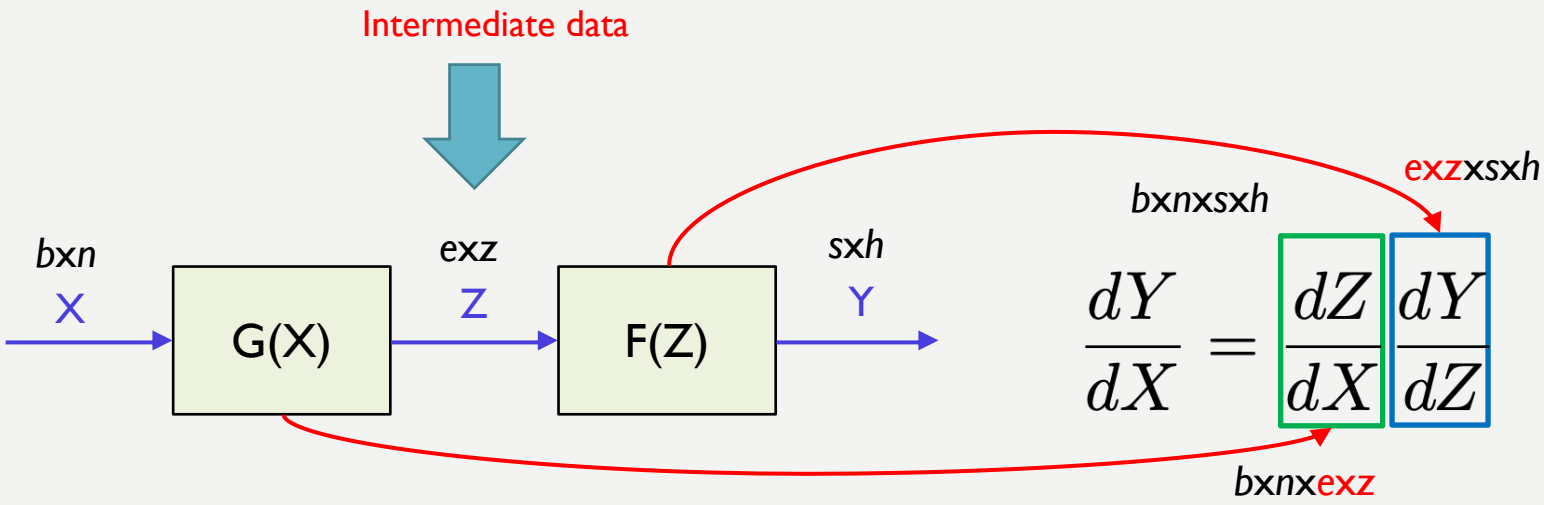
COMPUTATIONAL GRAPHS

- Let's consider computational nodes that are about vector functions
- Input and output data are now tensors
  - Check the following example: derivatives are combined in the **opposite order** with respect to the previous example(s), since they are now **tensors** and we assumed a **denominator layout**
  - They go from the *beginning of the path to the end of it*



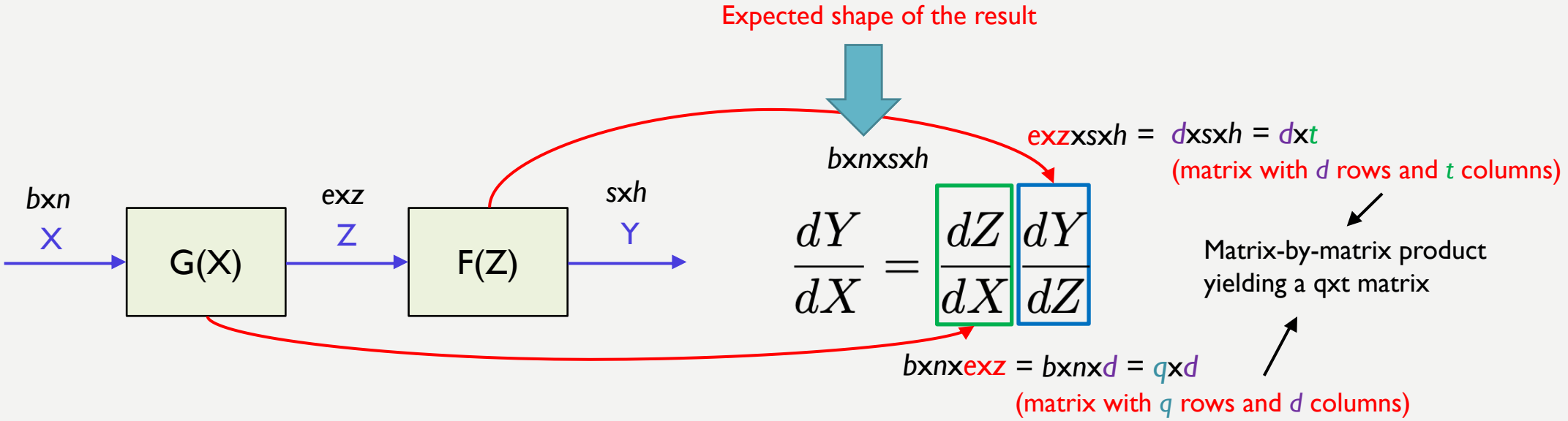
COMPUTATIONAL GRAPHS

- For **each pair** of nodes in path we have some **intermediate** data (*Z* in this example).
- The dimensions of these intermediate data are the ones on which the chain rule must "accumulate" the derivatives of the two nodes (so the dimensions of *Z* in this example, that are exz).



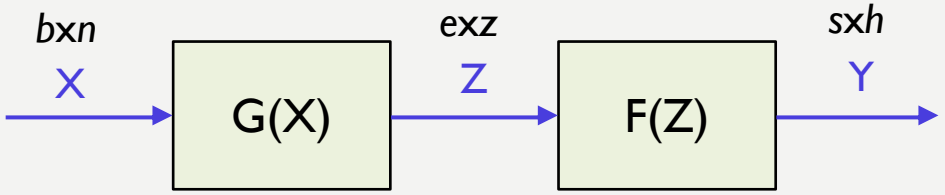
COMPUTATIONAL GRAPHS

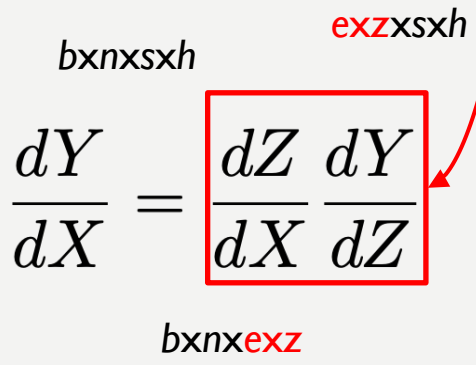
- In order to qualitatively understand the way the derivatives are "accumulated", reshape the tensor derivatives as shown in the example, then perform a matrix-by-matrix product
- Finally, we restore the expected shape of the result by another reshaping operation



COMPUTATIONAL GRAPHS

- Notice that when we apply the chain rule, we are performing new operations
- In turn, these new operations could be described with **their own** computational graph ☺





GOING BACK TO BACKPROGATION

- *The derivative of the objective function w.r.t. the weights W of the network is what is computed by means of Backpropagation*

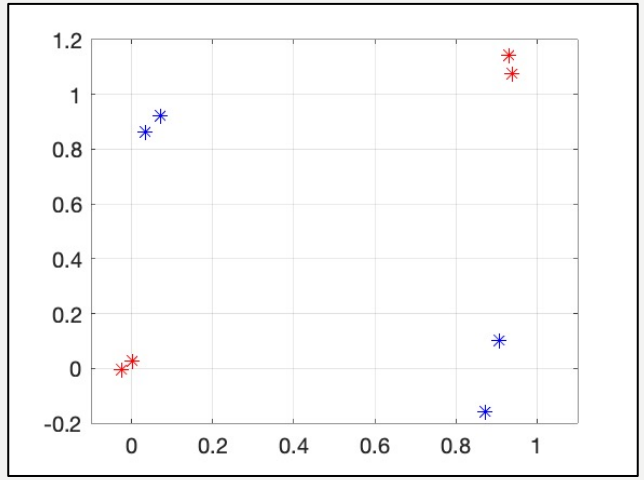
$$\frac{\partial \texttt{obj}(f(X,W),Y)}{\partial W}$$

X, Y	Training set (inputs, targets)
$f(X,W)$	Output of the network on the training set
$\texttt{obj}(f(X,W),Y)$	Loss function (objective)

- If we describe the **forward** operations performed by a neural network, up to the loss computation, using a **computational graph**, then we can apply the **chain rule** over all the possible paths connecting the weights of the network to the objective function
  - Summing up the results, we get the outcome of Backpropagation ☺
  - *Notice that what does matter is the use of the chain rule: the computational graph is just a formalism/tool we use to describe what is going on, it is not playing any other special roles*

BACKPROPAGATION FROM SCRATCH

- We consider a training set composed of 8 bidimensional examples (XOR problem)
  - 4 examples of class label 0 (blue), 4 examples of class label 1 (red)
  - Training examples are collected into matrix *X* *(one row per example, 8x2 matrix of floating-point numbers)*
  - Binary targets of the training examples are collected in vector **y** (*one target per example, 8 element-vector composed of zeros and ones*)

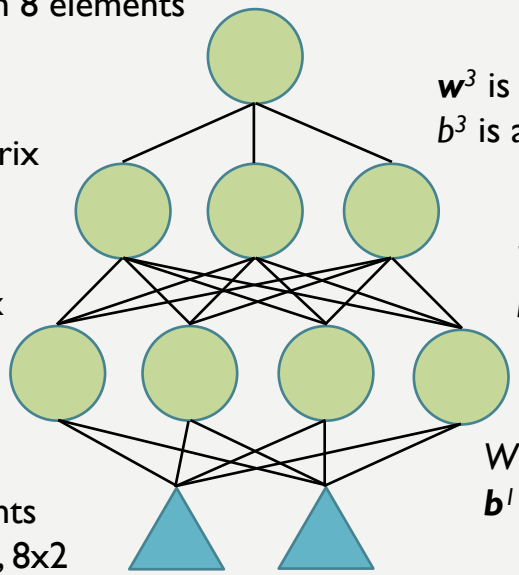


$X, \mathbf{y}$

- **Goal: train a neural network to learn from these data**
  - We will reimplement Backpropagation from scratch, using Numpy!
  - Batch-mode learning (forward on the whole training set, backward, …)

BACKPROPAGATION FROM SCRATCH

- We consider the following feed-forward network with **sigmoidal** activation functions
  - **Weights** are collected in $W^1, W^2, \mathbf{w}^3$, while **biases** (*not shown in the picture*) are collected in $\mathbf{b}^1, \mathbf{b}^2, b^3$



$\mathbf{o}^3$ is a vector with 8 elements

O^2 is a 8x3 matrix

O^1 is a 8x4 matrix

$\mathbf{w}^3$ is vector with 3 elements
b^3 is a scalar

W^2 is a 4x3 matrix
$\mathbf{b}^2$ is a vector with 3 elements

W^1 is a 2x4 matrix
$\mathbf{b}^1$ is a vector with 4 elements

Each input x is a vector of 2 elements
X collects all the training examples, 8x2

$$O^1 = \sigma(XW^1 + (\mathbf{b}^1\mathbf{1}^T)^T)$$

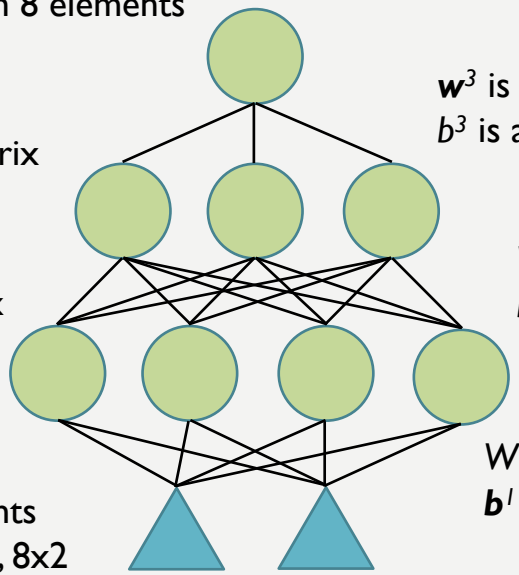
$$O^2 = \sigma(O^1W^2 + (\mathbf{b}^2\mathbf{1}^T)^T)$$

$$\mathbf{o}^3 = \sigma(O^2\mathbf{w}^3 + \mathbf{1}b^3)$$

$$\texttt{loss} = \|\mathbf{o}^3 - \mathbf{y}\|^2$$

BACKPROPAGATION FROM SCRATCH

- We consider the following feed-forward network with **sigmoidal** activation functions
  - **Weights** are collected in $W^1, W^2, \mathbf{w}^3$, while **biases** (*not shown in the picture*) are collected in $\mathbf{b}^1, \mathbf{b}^2, b^3$



$\mathbf{o}^3$ is a vector with 8 elements

O^2 is a 8x3 matrix

O^1 is a 8x4 matrix

$\mathbf{w}^3$ is vector with 3 elements
b^3 is a scalar

W^2 is a 4x3 matrix
$\mathbf{b}^2$ is a vector with 3 elements

W^1 is a 2x4 matrix
$\mathbf{b}^1$ is a vector with 4 elements

Each input x is a vector of 2 elements
X collects all the training examples, 8x2

Vectors of ones (8 elements)

$$O^1 = \sigma(XW^1 + (\mathbf{b}^1\mathbf{1}^T)^T)$$

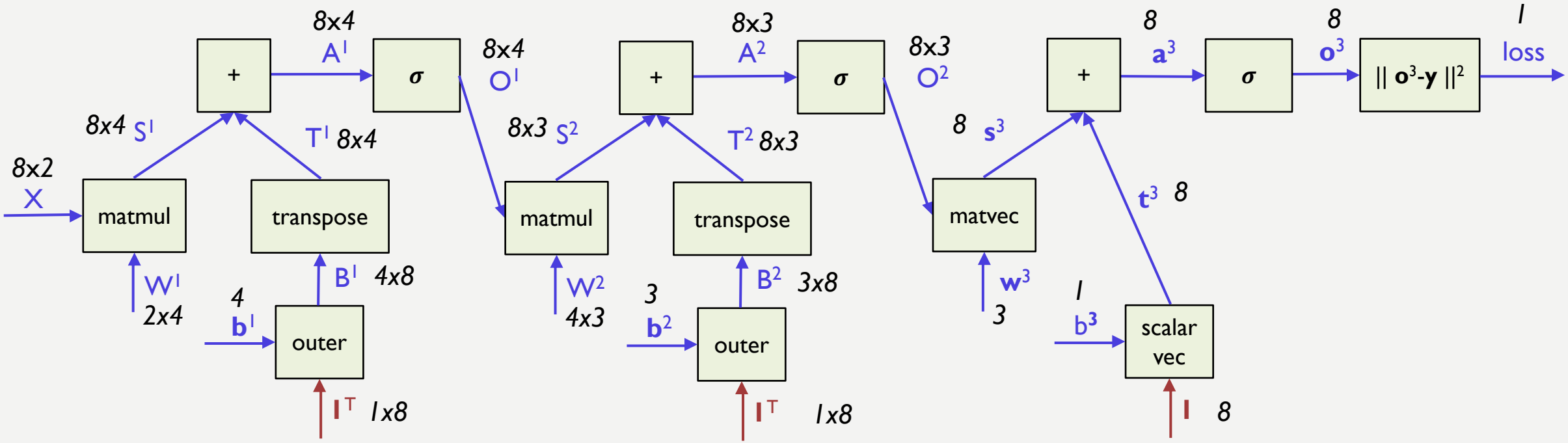
$$O^2 = \sigma(O^1W^2 + (\mathbf{b}^2\mathbf{1}^T)^T)$$

$$\mathbf{o}^3 = \sigma(O^2\mathbf{w}^3 + \mathbf{1}b^3)$$

$$\texttt{loss} = \|\mathbf{o}^3 - \mathbf{y}\|^2$$

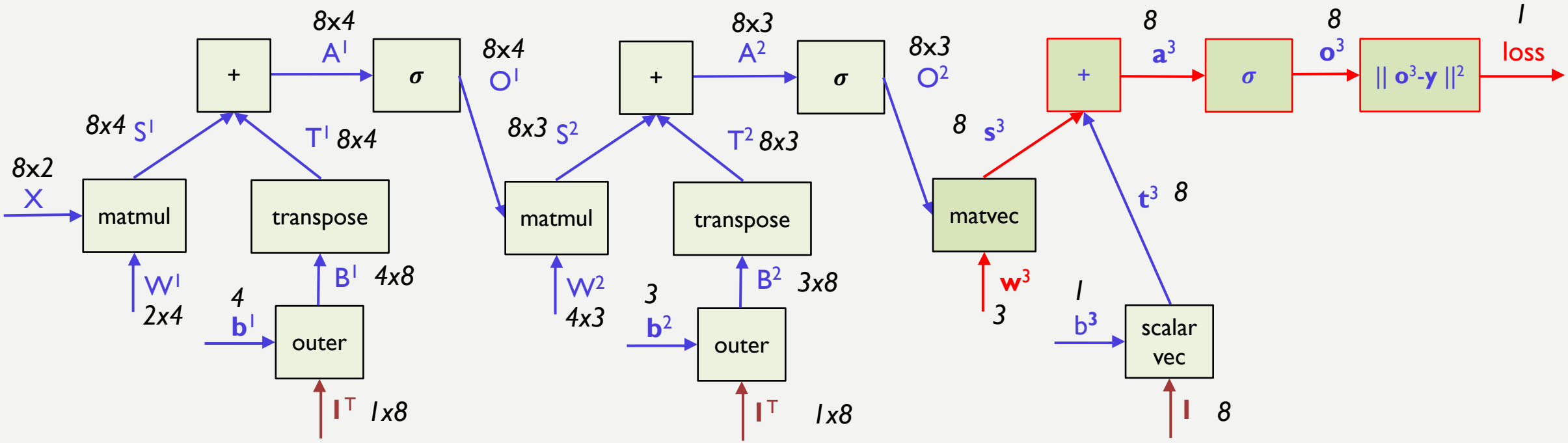
BACKPROPAGATION FROM SCRATCH

- Computational graph of the operations required to compute the loss (*forward*)



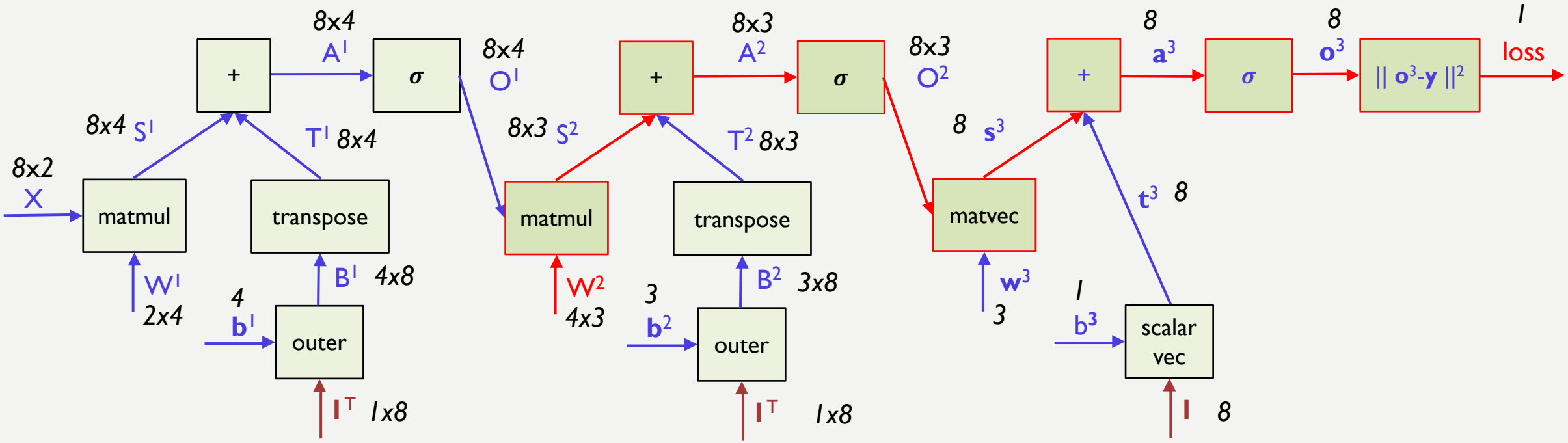
BACKPROPAGATION FROM SCRATCH

- Path that we must follow (in reverse order) to get the derivative of the loss w.r.t. to w^3



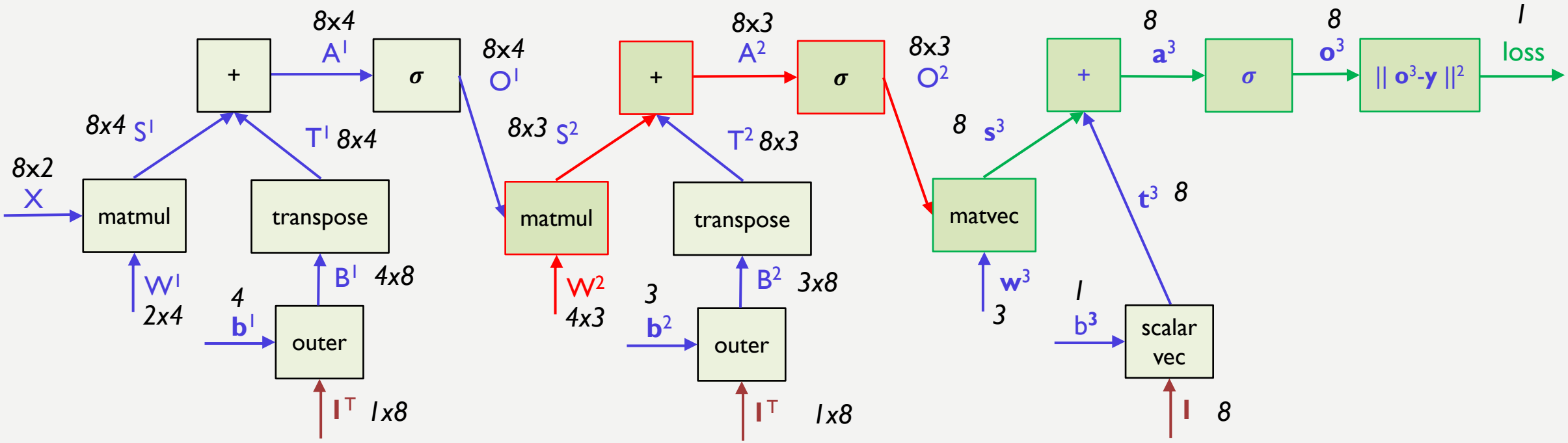
BACKPROPAGATION FROM SCRATCH

- Path that we must follow (in reverse order) to get the derivative of the loss w.r.t. to W^2



BACKPROPAGATION FROM SCRATCH

- The greenish portion of the path was already computed when computing the derivative w.r.t. $\boldsymbol{w}^3$, we can reuse some intermediate results. Oh, they are the already described Backprop **DELTAS!**



BACKPROPAGATION FROM SCRATCH

- Let's compute all the derivatives we need, in backward fashion, starting from $\boldsymbol{w}^3$
- We can recover the same elements/notation we introduced when describing Backpropagation

$$\underset{3}{\frac{\partial \mathtt{loss}}{\partial \mathbf{w}^3}} = \underbrace{\underbrace{\underset{3x8}{\frac{\partial \mathbf{s}^3}{\partial \mathbf{w}^3}} \underset{8x8}{\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3}}}_{\frac{\partial \mathbf{a}^3}{\mathbf{w}^3}} \underbrace{\underset{8x8}{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3}} \underset{8}{\frac{\partial \mathtt{loss}}{\partial \mathbf{o}^3}}}_{\delta^3}}_{3x8} \quad 8$$

Better compute them in this order!

BACKPROPAGATION FROM SCRATCH

- We apply the tensor derivatives we previously described…

$$\frac{\partial \texttt{loss}}{\partial \mathbf{w}^3} = \underbrace{\frac{\partial \mathbf{s}^3}{\partial \mathbf{w}^3}}_{\frac{\partial \mathbf{a}^3}{\mathbf{w}^3}} \frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3} \underbrace{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3} \frac{\partial \texttt{loss}}{\partial \mathbf{o}^3}}_{\delta^3}$$

3 | 3x8 | 8x8 | 8x8 | 8 | 8 | 3x8

$$\frac{\partial \texttt{loss}}{\partial \mathbf{o}^3} = 2(\mathbf{o}^3 - \mathbf{y})$$ vector

$$\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3} = \begin{bmatrix} \frac{\partial o_1^3}{\partial a_1^3} & \frac{\partial o_2^3}{\partial a_1^3} & \cdots & \frac{\partial o_8^3}{\partial a_1^3} \\ \frac{\partial o_1^3}{\partial a_2^3} & \frac{\partial o_2^3}{\partial a_2^3} & \cdots & \frac{\partial o_8^3}{\partial a_2^3} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial o_1^3}{\partial a_8^3} & \frac{\partial o_2^3}{\partial a_8^3} & \cdots & \frac{\partial o_8^3}{\partial a_8^3} \end{bmatrix} = \begin{bmatrix} \sigma'(a_1^3) & 0 & \dots & 0 \\ 0 & \sigma'(a_2^2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma'(a_8^8) \end{bmatrix}$$

derivative of the activation function w.r.t. its argument

diagonal matrix!

BACKPROPAGATION FROM SCRATCH

- When we chain two derivatives, we can simplify the computations exploiting the sparseness of the (tensors that store the) derivatives

$$\underset{3}{\frac{\partial \texttt{loss}}{\partial \mathbf{w}^3}} = \underbrace{\underbrace{\underset{3x8}{\frac{\partial \mathbf{s}^3}{\partial \mathbf{w}^3}}\underset{8x8}{\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3}}}_{\frac{\partial \mathbf{a}^3}{\mathbf{w}^3}}\underbrace{\underset{8x8}{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3}}\underset{8}{\frac{\partial \texttt{loss}}{\partial \mathbf{o}^3}}}_{\delta^3}}_{8}$$

$$\underset{8}{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3}\frac{\partial \texttt{loss}}{\partial \mathbf{o}^3}} = \sigma'\left(\mathbf{a}^3\right) \odot 2(\mathbf{o}^3 - \mathbf{y})$$

a diagonal matrix multplied by a vector can be written as an element-wise vector multiplication:
Hadamard product

The first operand of this product was originally supposed to be an 8x8 matrix, while it is now a vector with 8 elements (we exploited the structure/sparseness of the two operands)

$$\underset{8}{\delta^3} = \sigma'\left(\mathbf{a}^3\right) \odot 2(\mathbf{o}^3 - \mathbf{y})$$

Simplification like these will be frequently applied

BACKPROPAGATION FROM SCRATCH

- After having computed the last two terms, we can get the result we were looking for

$$\frac{\partial \texttt{loss}}{\partial \mathbf{w}^3} = \underbrace{\frac{\partial \mathbf{s}^3}{\partial \mathbf{w}^3}}_{\frac{\partial \mathbf{a}^3}{\mathbf{w}^3}} \frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3} \underbrace{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3} \frac{\partial \texttt{loss}}{\partial \mathbf{o}^3}}_{\delta^3}$$

3 — 3x8 — 8x8 — 8x8 — 8 — 8 — 3x8

$$\delta^3 = \sigma'\left(\mathbf{a}^3\right) \odot 2(\mathbf{o}^3 - \mathbf{y})$$

$$\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3} = \frac{\partial(\mathbf{s}^3 + \mathbf{t}^3)}{\partial \mathbf{s}^3} = I$$ (8x8)

$$\frac{\partial \mathbf{s}^3}{\partial \mathbf{w}^3} = \frac{\partial(O^2 \mathbf{w}^3)}{\partial \mathbf{w}^3} = (O^2)^T$$ (3x8)

$$\frac{\partial \texttt{loss}}{\partial \mathbf{w}^3} = (O^2)^T \delta^3 = (O^2)^T \left(\sigma'\left(\mathbf{a}^3\right) \odot 2\left(\mathbf{o}^3 - \mathbf{y}\right)\right)$$

BACKPROPAGATION FROM SCRATCH

- What about the bias term b^3?

$$\underset{1}{\frac{\partial \texttt{loss}}{\partial b^3}} = \underbrace{\underbrace{\underset{1x8}{\frac{\partial \mathbf{t}^3}{\partial b^3}}\underset{8x8}{\frac{\partial \mathbf{a}^3}{\partial \mathbf{t}^3}}}_{\frac{\partial \mathbf{a}^3}{b^3}}\underbrace{\underset{8x8}{\frac{\partial \mathbf{o}^3}{\partial \mathbf{a}^3}}\underset{8}{\frac{\partial \texttt{loss}}{\partial \mathbf{o}^3}}}_{\delta^3}}_{1x8}$$

Already computed

$$\frac{\partial \mathbf{a}^3}{\partial \mathbf{t}^3} = \frac{\partial(\mathbf{s}^3 + \mathbf{t}^3)}{\partial \mathbf{t}^3} = \overset{8x8}{I}$$

$$\frac{\partial \mathbf{t}^3}{\partial b^3} = \frac{\partial(b^3 \mathbf{1})}{\partial b^3} = \overset{1x8}{\mathbf{1}^T}$$

$$\frac{\partial \texttt{loss}}{\partial b^3} = \mathbf{1}^T \delta^3 = \mathbf{1}^T \left(\sigma'\left(\mathbf{a}^3\right) \odot 2\left(\mathbf{o}^3 - \mathbf{y}\right)\right)$$

BACKPROPAGATION FROM SCRATCH

- Let's keep going backward, derivative w.r.t. W^2

$$\frac{\partial \texttt{loss}}{\partial W^2} = \underbrace{\frac{\partial S^2}{\partial W^2}\frac{\partial A^2}{\partial S^2}}_{\frac{\partial A^2}{\partial W^2}}\underbrace{\frac{\partial O^2}{\partial A^2}\frac{\partial \mathbf{s}^3}{\partial O^2}\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3}\delta^3}_{\Delta^2}$$

4x3 | 4x3x8x3 | 8x3x8x3 | 8x3x8 | 8x8 | 8 | 8x3

4x3x8x3

$$\delta^3 = \sigma'\left(\mathbf{a}^3\right) \odot 2(\mathbf{o}^3 - \mathbf{y})$$

Already computed ☺

$$\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3} = \frac{\partial(\mathbf{s}^3 + \mathbf{t}^3)}{\partial \mathbf{s}^3} = I$$

It is just the **identity matrix**, we can forget about it / remove it

BACKPROPAGATION FROM SCRATCH

- Let's keep going backward, derivative w.r.t. of W^2

$$\frac{\partial \texttt{loss}}{\partial W^2} = \underbrace{\frac{\partial S^2}{\partial W^2}}_{\frac{\partial A^2}{\partial W^2}} \underbrace{\frac{\partial A^2}{\partial S^2} \frac{\partial O^2}{\partial A^2} \frac{\partial \mathbf{s}^3}{\partial O^2}}_{\Delta^2} \underbrace{\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3}}_{\delta^3}$$

4x3 | 4x3x8x3 | 8x3x8x3 | 8x3x8 | 8x8 | 8

4x3x8x3 | 8x3

$$\frac{\partial \mathbf{s}^3}{\partial O^2} = \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O^2} = \begin{bmatrix} \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{11}} & \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{12}} & \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{13}} \\ \vdots & \vdots & \vdots \\ \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{81}} & \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{82}} & \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O_{83}} \end{bmatrix}$$

8x3x8

You can read this result (tensor) as follows:
each element of the 8x3 matrix above is a vector with 8 elements

Let's see the whole tensor in the next slide!
(be ready)

BACKPROPAGATION FROM SCRATCH

- The tensor is actually very **sparse**!

sparse vector

$$\frac{\partial \mathbf{s}^3}{\partial O^2} = \frac{\partial \left(O^2 \mathbf{w}^3\right)}{\partial O^2} = \begin{bmatrix} [w_1^3,0,0,0,0,0,0,0]^T & [w_2^3,0,0,0,0,0,0,0]^T & [w_3^3,0,0,0,0,0,0,0]^T \\ [0,w_1^3,0,0,0,0,0,0]^T & [0,w_2^3,0,0,0,0,0,0]^T & [0,w_3^3,0,0,0,0,0,0]^T \\ \vdots & \vdots & \vdots \\ [0,0,0,0,0,0,0,w_1^3]^T & [0,0,0,0,0,0,0,w_2^3]^T & [0,0,0,0,0,0,0,w_3^3]^T \end{bmatrix}$$

- Looking back at the previous slide, notice that we are now going to multiply it by the vector 'delta'

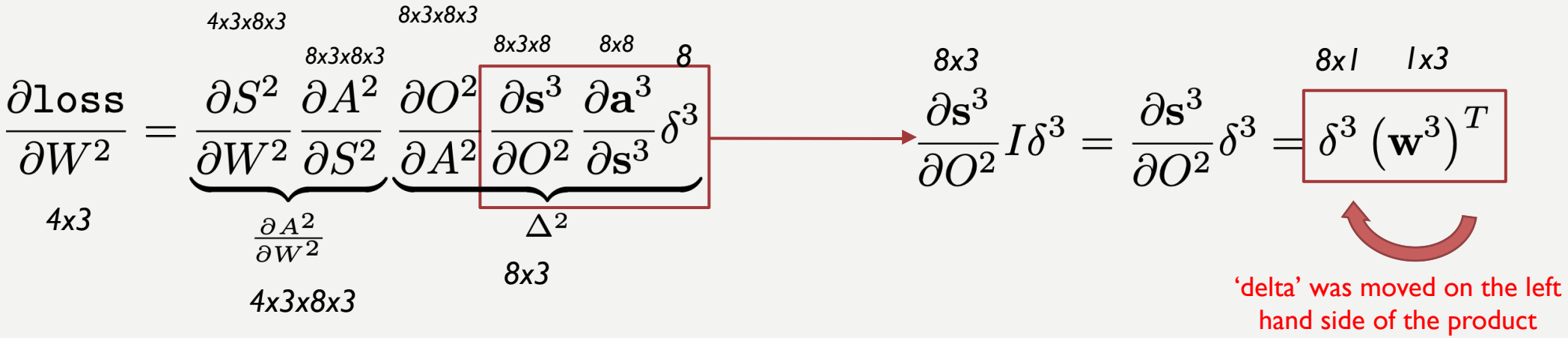
$$\dots \frac{\partial \mathbf{s}^3}{\partial O^2} I\delta^3 = \dots \frac{\partial \mathbf{s}^3}{\partial O^2} \delta^3$$

This is the formula in the previous slide, where some elements on the left portion of the formula were replaced by dots, for better readability

  - This means that each of the sparse vectors above will be multiplied (dot product) by 'delta'
  - As a result, **only one component of 'delta' will be preserved after each dot product**
  - We get a matrix in which the element in position *i,j* is $\delta_i^3 w_j^3$

BACKPROPAGATION FROM SCRATCH

- We can write the result of the just described computation in a more compact manner!
  - No more big-sparse tensors ☺



BACKPROPAGATION FROM SCRATCH

- The remaining terms yield a similar structure to the one we already experienced in the layer above
  - Now we have matrices, a bit more generic case

$$\frac{\partial \texttt{loss}}{\partial W^2} = \underbrace{\frac{\partial S^2}{\partial W^2}}_{\frac{\partial A^2}{\partial W^2}} \underbrace{\frac{\partial A^2}{\partial S^2} \frac{\partial O^2}{\partial A^2} \frac{\partial \mathbf{s}^3}{\partial O^2} \frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3} \delta^3}_{\Delta^2}$$

4x3 | 4x3x8x3 | 8x3x8x3 | 8x3x8 | 8x8 | 8 | 8x3 | 4x3x8x3

$$\frac{\partial O^2}{\partial A^2} = \frac{\partial \left(\sigma(A^2)\right)}{\partial A^2}$$

It is always zero except for those elements whose indices are *(i,j,i,j)*.

We can apply again the 'trick' of the Hadamard product with the matrix $\underset{8x3}{\sigma'(A^2)}$

$$\frac{\partial S^2}{\partial W^2} = \frac{\partial \left(O^1 W^2\right)}{\partial W^2} = \ldots sparse\ tensor \ldots = \underset{4x8}{\left(O^1\right)^T}$$

It has no effects, we can forget about it (it is a tensor composed of identity matrices)

$$\frac{\partial \texttt{loss}}{\partial W^2} = (O^1)^T \Delta^2 = (O^1)^T \left(\sigma'\left(A^2\right) \odot \delta^3 (\mathbf{w}^3)^T\right)$$

BACKPROPAGATION FROM SCRATCH

- What about the bias vector $\boldsymbol{b}^2$?

$$\frac{\partial \texttt{loss}}{\partial \mathbf{b}^2} = \underbrace{\frac{\partial B^2}{\partial \mathbf{b}^2}}_{3x3x8} \underbrace{\frac{\partial T^2}{\partial B^2}}_{3x8x8x3} \underbrace{\frac{\partial A^2}{\partial T^2}}_{8x3x8x3} \underbrace{\frac{\partial O^2}{\partial A^2}}_{8x3x8x3} \underbrace{\frac{\partial \mathbf{s}^3}{\partial O^2}}_{8x3x8} \underbrace{\frac{\partial \mathbf{a}^3}{\partial \mathbf{s}^3}}_{8x8} \underbrace{\delta^3}_{8}$$

$\frac{\partial A^2}{\partial \mathbf{b}^2}$: 3x8x3

Δ^2 : 8x3

3

It has no effects, we can forget about it (it is a tensor composed of identity matrices)

Already computed

$$\frac{\partial T^2}{\partial B^2} = \frac{\partial \left(B^2\right)^T}{\partial B^2}$$

It is a sparse tensor with a few elements equal to one. The effect of multiplying this tensor with the following product of terms in the chain rule is of transposing the outcome of such product

$$\frac{\partial T^2}{\partial B^2} \times Z = Z^T$$

$$\frac{\partial B^2}{\partial \mathbf{b}^2} = \frac{\partial \left(\mathbf{b}^2 \mathbf{1}^T\right)}{\partial \mathbf{b}^2}$$

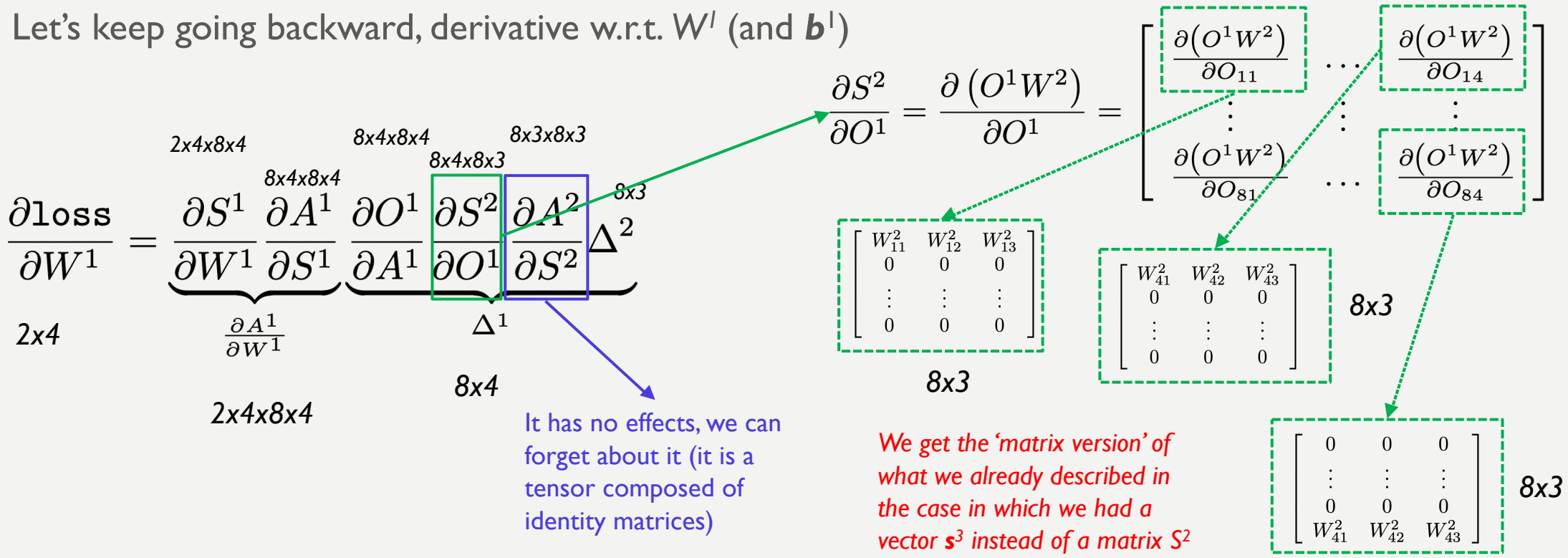
The effect of multiplying this tensor with the following product of terms in the chain rule is of post-multiplying by a vector of ones

$$\frac{\partial B^2}{\partial \mathbf{b}^2} \times U = U\mathbf{1}$$

$$\boxed{\frac{\partial \texttt{loss}}{\partial \mathbf{b}^2} = \left(\Delta^2\right)^T \mathbf{1}}$$

BACKPROPAGATION FROM SCRATCH

- Let's keep going backward, derivative w.r.t. W^l (and $\boldsymbol{b}^l$)



BACKPROPAGATION FROM SCRATCH

- Let's keep going backward, derivative w.r.t. W^l

$$\frac{\partial \texttt{loss}}{\partial W^1} = \underbrace{\frac{\partial S^1}{\partial W^1}}_{\frac{\partial A^1}{\partial W^1}} \underbrace{\frac{\partial A^1}{\partial S^1} \frac{\partial O^1}{\partial A^1} \frac{\partial S^2}{\partial O^1} \frac{\partial A^2}{\partial S^2} \Delta^2}_{\Delta^1}$$

2x4 | 2x4x8x4 | 8x4x8x4 | 8x4x8x3 | 8x3x8x3 | 8x3 | 8x4 | 2x4x8x4

$$\frac{\partial S^2}{\partial O^1} \Delta^2 = \boxed{\Delta^2 \left(W^2\right)^T} \quad 8x4$$

'Delta' was moved on the left hand side of the product

We already discussed how to compute the other derivatives in the case of the layer above, so we can go straight to the final result
(the case of the bias term is fully equivalent to what we already discussed)

$$\boxed{\frac{\partial \texttt{loss}}{\partial W^1} = X^T \Delta^1 = X^T \left(\sigma'\left(A^1\right) \odot \Delta^2 \left(W^2\right)^T\right)}$$

$$\boxed{\frac{\partial \texttt{loss}}{\partial \mathbf{b}^1} = \left(\Delta^1\right)^T \mathbf{1}}$$

BACKPROPAGATION FROM SCRATCH

- **Extended summary**
  - Copy & paste of the previous formulas

In our example we considered a single neuron in the last layer.
We can make this general, and consider multiple output neurons…

$$\frac{\partial \texttt{loss}}{\partial \mathbf{w}^3} = (O^2)^T \delta^3 = (O^2)^T \left(\sigma'\left(\mathbf{a}^3\right) \odot 2\left(\mathbf{o}^3 - \mathbf{y}\right)\right)$$

$$\frac{\partial \texttt{loss}}{\partial W^2} = (O^1)^T \Delta^2 = (O^1)^T \left(\sigma'\left(A^2\right) \odot \delta^3 (\mathbf{w}^3)^T\right)$$

$$\frac{\partial \texttt{loss}}{\partial W^1} = X^T \Delta^1 = X^T \left(\sigma'\left(A^1\right) \odot \Delta^2 \left(W^2\right)^T\right)$$

$$\frac{\partial \texttt{loss}}{\partial b^3} = \mathbf{1}^T \delta^3 = \mathbf{1}^T \left(\sigma'\left(\mathbf{a}^3\right) \odot 2\left(\mathbf{o}^3 - \mathbf{y}\right)\right)$$

$$\frac{\partial \texttt{loss}}{\partial \mathbf{b}^2} = \left(\Delta^2\right)^T \mathbf{1}$$

$$\frac{\partial \texttt{loss}}{\partial \mathbf{b}^1} = \left(\Delta^1\right)^T \mathbf{1}$$

BACKPROPAGATION FROM SCRATCH

- **Compact summary**
  - Compute/update deltas
  - Compute derivatives w.r.t. weights and biases

num_examples x num_neurons_layer_L

$$\Delta^L = \sigma'\left(A^L\right) \odot \frac{\partial \texttt{loss}}{\partial O^L}$$

num_examples x num_neurons_layer_l

$$\Delta^\ell = \sigma'\left(A^\ell\right) \odot \Delta^{\ell+1}\left(W^{\ell+1}\right)^T, \qquad 1 \leq \ell < L$$

num_neurons_layer_l-1 x num_neurons_layer_l

$$\frac{\partial \texttt{loss}}{\partial W^\ell} = \left(O^{\ell-1}\right)^T \Delta^\ell, \qquad 1 \leq \ell \leq L,\ O^0 = X$$

num_neurons_layer_l

$$\frac{\partial \texttt{loss}}{\partial \mathbf{b}^\ell} = \left(\Delta^\ell\right)^T \mathbf{1}, \qquad 1 \leq \ell \leq L$$

BACKPROPAGATION FROM SCRATCH

- **Some take-home hints**
  - *If f(Z) is a function that operates element-wise on its argument Z (activation functions)*

$$\frac{\partial f(Z)}{\partial Z}\frac{\partial \texttt{loss}}{f(Z)} = f'(Z) \odot \frac{\partial \texttt{loss}}{f(Z)}$$

  - *If we have a tensor times another tensor, differentiated w.r.t. the second one*

$$\frac{\partial (OW)}{\partial W}\frac{\partial \texttt{loss}}{\partial W} = O^T \frac{\partial \texttt{loss}}{\partial W}$$

  - *If we have a tensor times another tensor differentiated w.r.t. to the first one (**swap order**!)*

$$\frac{\partial (OW)}{\partial O}\frac{\partial \texttt{loss}}{\partial (OW)} = \frac{\partial \texttt{loss}}{\partial (OW)} W^T$$

CODE BREAK: IMPLEMENTING NEURNAL NETWORKS & BACKPROP

- Let's have a look at the code!
  - NumPy-based implementation of
    - Feed-forward neural networks
    - Backpropagation