Backpropagation lecture notes

Nicolò Navarin

The notation from the "Deep Learning" book follows (we will try to simplify this notation)

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$

Notation in this lecture notes: variables and constants are in lower case, vectors are bold, matrices are uppercase bold. We will tend to use the partial derivative/Jacobian matrix notation from the book "Mathematics for Machine Learning".

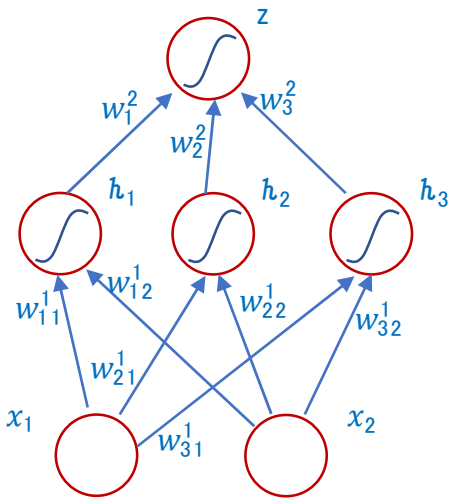
Recall:

$$\sigma(net) = \frac{1}{1 + e^{-net}} \qquad \sigma'^{(net)} = \frac{d\,\sigma(net)}{d\,net} = \sigma(net)(1 - \sigma(net))$$

Chain Rule:

$$\frac{d\,f(g(x))}{d\,x} = \frac{d\,f(g(x))}{d\,g(x)} \frac{d\,g(x)}{d\,x}$$

We are given a training set $Tr = \{(\boldsymbol{x}^{(1)}, t^{(1)}), \dots, (\boldsymbol{x}^{(N_{Tr})}, t^{(N_{Tr})})\}$ and a network:



Forward pass

computes z and all the required variables for each example $\boldsymbol{x}^{(i)}$. The variables to be stored will be clear when computing the derivatives.
In the following we omit the superscript (i) indicating the specific training example for ease of notation.

$a_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2 + b_1^1, a_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2 + b_2^1,\ a_3^1 = w_{31}^1 x_1 + w_{32}^1 x_2 + b_3^1$
$\mathrm{h}_1 = \sigma(a_1^1),\ \mathrm{h}_2 = \sigma(a_2^1), \mathrm{h}_3 = \sigma(a_3^1)$
$a^2 = w_1^2 \mathrm{h}_1 + w_2^2 \mathrm{h}_2 + b^2$
$z = \sigma(a^2)$

Matrix notation:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{W}^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \\ w_{31}^1 & w_{32}^1 \end{bmatrix}, b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}, \boldsymbol{a}^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix} = \boldsymbol{W}^1 \boldsymbol{x} + \boldsymbol{b}^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \\ w_{31}^1 & w_{32}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \sigma(\boldsymbol{a}^1), \qquad \boldsymbol{W}^2 = \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \end{bmatrix},$$

$$z = (\boldsymbol{w}^2)^T \boldsymbol{h} + b^2 = \begin{bmatrix} w_1^2 & w_2^2 & w_3^2 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = w_1^2 h_1 + w_2^2 h_2 + w_3^2 h_3 + b^2$$

Gradient Computation

Let us consider the MSE loss $J = \frac{1}{2}\frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)^2$

Let's compute the gradient with respect to the output weights w_1^2 (same as linear regression)

$\frac{\partial J}{\partial w_1^2} = \frac{\partial}{\partial w_1^2}\frac{1}{2N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)^2 =$ Derivative is linear: derivative of sum is the sum of derivatives

$= \frac{1}{2N_{Tr}}\sum_{p\in Tr}\frac{\partial}{\partial w_1^2}\left(t^{(p)} - z^{(p)}\right)^2 =$ Power rule: $\frac{d\, f(x)^n}{dx} = n\left(f(x)\right)^{n-1} f'(x)$

$= \frac{1}{2N_{Tr}}\sum_{p\in Tr} 2\left(t^{(p)} - z^{(p)}\right)\frac{\partial}{\partial w_1^2}\left(t^{(p)} - z^{(p)}\right)$ $\quad z^{(p)} = \sigma(w_1^2 h_1 + w_2^2 h_2 + w_3^2 h_3 + b^2) = \sigma(\,(\boldsymbol{w}^2)^T \boldsymbol{h}^p)$

$= \frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\frac{\partial}{\partial w_1^2}\left(t^{(p)} - \sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^p\right)\right)$ Sum rule. $\frac{\partial\, t^{(p)}}{\partial w_1^2} = 0$

$\nabla_{z^{(p)}} J$ $\quad f'(a^2)$ $\quad$ chain rule: $-\frac{\partial\sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)}{\partial w_1^2} = -\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)h_1^{(p)}$

$= -\frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^p\right)h_1^p = \frac{dJ}{dz}\frac{dz}{da^2}\frac{da^2}{dw_1^2}$

Computed during backward pass $\quad$ Computed during forward pass

For sigmoid activation function

$$\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right) = \frac{d\,\sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)}{d\,(\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}} = \sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)\left(1 - \sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)\right)$$

$\frac{\partial J}{\partial w_1^2} = -\frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^p\right)\left(1 - \sigma\left((\boldsymbol{w}^2)^T \boldsymbol{h}^p\right)\right)h_1^p$

Similar procedure for w_2^2, w_3^2 and for the bias values

Let us now compute the gradient w.r.t. the weights of the first layer, e.g. w_1^1

$\frac{\partial J}{\partial w_{11}^1} = \frac{\partial}{\partial w_1^2}\frac{1}{2N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)^2 =$ Sum rule

$= \frac{1}{2N_{Tr}}\sum_{p\in Tr}\frac{\partial}{\partial w_1^1}\left(t^{(p)} - z^{(p)}\right)^2 =$ Power rule and $\frac{\partial\, t^{(p)}}{\partial w_1^1} = 0$ and sum rule

$= \frac{1}{2N_{Tr}}\sum_{p\in Tr} 2\left(t^{(p)} - z^{(p)}\right)\frac{\partial}{\partial w_{11}^1}\left(-z^{(p)}\right)$ $= z^{(p)} = \sigma(\,(\boldsymbol{w}^2)^T \boldsymbol{h}^p)$ and chain rule

$= -\frac{1}{2N_{Tr}}\sum_{p\in Tr} 2\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)\frac{\partial}{\partial w_{11}^1}\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)$ $(\boldsymbol{w}^2)^T \boldsymbol{h}^p = w_1^2 h_1 + w_2^2 h_2 + w_3^2 h_3 + b^2$

$= -\frac{1}{2N_{Tr}}\sum_{p\in Tr} 2\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)\frac{\partial}{\partial w_{11}^1}w_1^2 h_1^{(p)} + \frac{\partial}{\partial w_{11}^1}w_2^2 h_2^{(p)} + \frac{\partial}{\partial w_{11}^1}w_3^2 h_3^{(p)} + \frac{\partial}{\partial w_{11}^1}b^2$

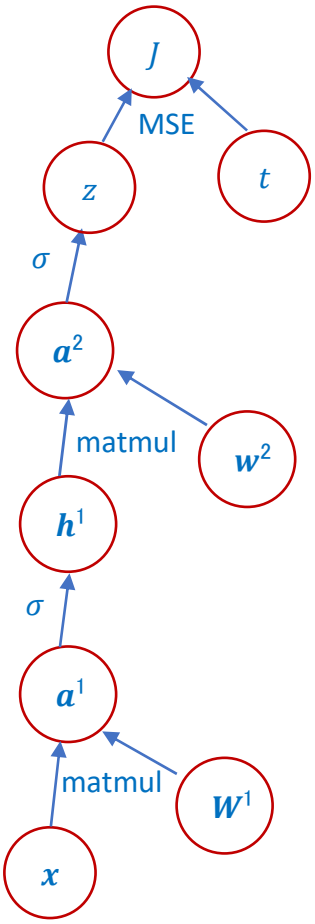
=0 $\quad$ Product rule + $h_1^p = \sigma(\boldsymbol{w}_1^1 \boldsymbol{x} + b_1^1)$ $\quad [w_{11}^1 \quad w_{12}^1]$

$= \frac{1}{-N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)w_1^2\frac{\partial}{\partial w_{11}^1}\sigma\left(\boldsymbol{w}_1^1 \boldsymbol{x}^{(\boldsymbol{p})} + b_1^1\right) =$ Chain rule

$= -\frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)w_1^2\sigma'\left(\boldsymbol{w}_1^1 \boldsymbol{x}^{(\boldsymbol{p})} + b_1^1\right)\frac{\partial}{\partial w_{11}^1}w_{11}^1 x_1^{(p)} + \frac{\partial}{\partial w_{11}^1}w_{12}^1 x_2^{(p)}$

$+ \frac{\partial}{\partial w_{11}^1}b_1^1 =$ $\quad$ =0

$= -\frac{1}{N_{Tr}}\sum_{p\in Tr}\left(t^{(p)} - z^{(p)}\right)\sigma'\left((\boldsymbol{w}^2)^T \boldsymbol{h}^{(p)}\right)w_1^2\sigma'\left(\boldsymbol{w}_1^1 \boldsymbol{x}^{(\boldsymbol{p})} + b_1^1\right)x_1^{(p)}$



Gradient Computation in matrix notation

Let us from now on consider a single example for simplicity.
Let's compute the gradient with respect to the output weights **vector** $\boldsymbol{w}^2$

$$\frac{\partial J}{\partial \boldsymbol{w}^2} = \nabla_{\boldsymbol{w}^2} J = \frac{\partial J}{\partial z}\frac{dz}{da^2}\frac{\partial a^2}{\partial \boldsymbol{w}^2} = \nabla_z J \sigma'(a^2)\, \nabla_{\boldsymbol{w}^2} a^2 = [-(t-z)]\left[\sigma(a^2)\left(1-\sigma(a^2)\right)\right]\nabla_{\boldsymbol{w}^2} a^2 =$$
$$= [-(t-z)]\left[\sigma(a^2)\left(1-\sigma(a^2)\right)\right][h_1 \quad h_2 \quad h_3] = [-(t-z)]\left[\sigma(a^2)\left(1-\sigma(a^2)\right)\right]\boldsymbol{h}^T$$

Notice the shape of $\frac{\partial J}{\partial \boldsymbol{w}^2}$. Since we defined the gradients as row vectors, we expect $\frac{\partial J}{\partial \boldsymbol{w}^2}$ to have a similar shape to $\boldsymbol{w}^2$. Actually, it should have the same shape as $(\boldsymbol{w}^2)^T$, so live in $\mathbb{R}^{1\times 3}$. This is obtained since a^2 is a number and $\boldsymbol{w}^2$ is a vector in $\mathbb{R}^3$, so for our definition of gradient $\nabla_{\boldsymbol{w}^2} a^2$ have to be a row vector of size $\mathbb{R}^{1\times 3}$.

Let's now compute the gradient with respect to the hidden weights **matrix** $\boldsymbol{W}^1$.
We will require the computation of the gradient of vectors with respect to matrices, please see

$$\mathbb{R} \quad \mathbb{R} \quad \mathbb{R}^{1\times 3} \quad \mathbb{R}^{3\times 3} \quad \mathbb{R}^{3\times(3\times 2)}$$
$$\frac{\partial J}{\partial \boldsymbol{W}^1} = \frac{\partial J}{\partial z}\frac{dz}{da^2}\frac{\partial a^2}{\partial \boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{a}^1}\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1}$$

Let us study the dimensions of each term.

$\frac{\partial J}{\partial z} \in \mathbb{R}$ since both J and z are numbers.

$\frac{dz}{da^2} \in \mathbb{R}$ since both z and a^2 are numbers. This is the derivative of the output after applying the output activation function w.r.t. the output pre-activation.

$\frac{\partial a^2}{\partial \boldsymbol{h}} \in \mathbb{R}^{1\times 3}$ is a row vector since a^2 is a number and $\boldsymbol{h}$ is a vector. In particular, it will be defined as:

$$\frac{\partial a^2}{\partial \boldsymbol{h}} = \begin{bmatrix} \frac{\partial a^2}{\partial h_1} & \frac{\partial a^2}{\partial h_2} & \frac{\partial a^2}{\partial h_3} \end{bmatrix} = [w_1^2 \quad w_2^2 \quad w_3^2]$$

$\frac{d\boldsymbol{h}}{d\boldsymbol{a}^1} \in \mathbb{R}^{3\times 3}$ is a (Jacobian) matrix since both $\boldsymbol{h}$ and $\boldsymbol{a}^1$ are vectors. This matrix however has a particularity: it is diagonal since h_i only depends on a_i^1 (pointwise non-linearity). Thus:

$$\frac{d\boldsymbol{h}}{d\boldsymbol{a}^1} = \begin{bmatrix} \nabla_{\mathbf{a}^1} h_1 \\ \nabla_{\mathbf{a}^1} h_1 \\ \nabla_{\mathbf{a}^1} h_1 \end{bmatrix} = \begin{bmatrix} \frac{\partial \boldsymbol{h}}{\partial a_1^1} & \frac{\partial \boldsymbol{h}}{\partial a_2^1} & \frac{\partial \boldsymbol{h}}{\partial a_3^1} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial a_1^1} & \frac{\partial h_1}{\partial a_2^1} & \frac{\partial h_1}{\partial a_3^1} \\ \frac{\partial h_2}{\partial a_1^1} & \frac{\partial h_2}{\partial a_2^1} & \frac{\partial h_2}{\partial a_3^1} \\ \frac{\partial h_3}{\partial a_1^1} & \frac{\partial h_3}{\partial a_2^1} & \frac{\partial h_3}{\partial a_3^1} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial a_1^1} & 0 & 0 \\ 0 & \frac{\partial h_2}{\partial a_2^1} & 0 \\ 0 & 0 & \frac{\partial h_3}{\partial a_3^1} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma'(a_1^1) & 0 & 0 \\ 0 & \sigma'(a_2^1) & 0 \\ 0 & 0 & \sigma'(a_3^1) \end{bmatrix}$$

Let us now focus on the term $\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1}$. $\boldsymbol{a}^1 \in \mathbb{R}^{\mathbf{3}}$ is a vector of size 3. $\boldsymbol{W}^1 \in \mathbb{R}^{\mathbf{3}\times\mathbf{2}}$ is the matrix of weights of the first layer. $\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1}$ is then a <u>tensor</u> in $\mathbb{R}^{\mathbf{3}\times(\mathbf{3}\times\mathbf{2})}$.

In fact, by definition, the gradient is the collection of partial derivatives:

$$\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} \frac{\partial a_1^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_2^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_3^1}{\partial \boldsymbol{W}^1} \end{bmatrix}, \text{ where } \frac{\partial a_i^1}{\partial \boldsymbol{W}^1} \in \mathbb{R}^{1\times(3\times 2)}$$

Let us write down explicitly $a_i^1 = w_{i1}^1 x_1 + w_{i2}^1 x_2 + b_i^1$

We know that $\frac{\partial a_i^1}{\partial \boldsymbol{W}_{ij}^1} = x_j$, that is $\frac{\partial a_i^1}{\partial \boldsymbol{W}_{i1}^1} = x_1$ and $\frac{\partial a_i^1}{\partial \boldsymbol{W}_{i2}^1} = x_2$

We can also compute the partial derivative of a_i^1 with respect to a row of $\boldsymbol{W}^1$, that is:

$\frac{\partial a_i^1}{\partial \boldsymbol{W}_{i,:}^1} = \boldsymbol{x}^T \in \mathbb{R}^{1\times 1\times 2}$. Moreover, the tensor is sparce since $\frac{\partial a_i^1}{\partial \boldsymbol{W}_{k\neq i,:}^1} = \mathbf{0}^T \in \mathbb{R}^{1\times 1\times 2}$

Let us now look more closely at $\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1}$

$$\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} \frac{\partial a_1^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_2^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_3^1}{\partial \boldsymbol{W}^1} \end{bmatrix}, \text{ and } \frac{\partial a_i^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} \frac{\partial a_i^1}{\partial W_{11}^1} & \frac{\partial a_i^1}{\partial W_{12}^1} \\ \frac{\partial a_i^1}{\partial W_{21}^1} & \frac{\partial a_i^1}{\partial W_{22}^1} \\ \frac{\partial a_i^1}{\partial W_{31}^1} & \frac{\partial a_i^1}{\partial W_{32}^1} \end{bmatrix} = \begin{bmatrix} \frac{\partial a_i^1}{\partial \boldsymbol{W}_{1,:}^1} \\ \frac{\partial a_i^1}{\partial \boldsymbol{W}_{2,:}^1} \\ \frac{\partial a_i^1}{\partial \boldsymbol{W}_{3,:}^1} \end{bmatrix}$$

Then we have that

$$\frac{\partial a_1^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} x_1 & x_2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \frac{\partial a_2^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} 0 & 0 \\ x_1 & x_2 \\ 0 & 0 \end{bmatrix} \text{ and } \frac{\partial a_3^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ x_1 & x_2 \end{bmatrix}$$

We can now try start to put things together and compute our gradients:

$$\frac{dJ}{d\boldsymbol{W}^1} = \frac{dJ}{dz}\frac{dz}{da^2}\frac{da^2}{d\boldsymbol{h}}\frac{d\boldsymbol{h}}{d\boldsymbol{a}^1}\frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1}$$

$$\frac{dJ}{d\boldsymbol{W}^1} = \nabla_z J \sigma'(a^2)\, \nabla_{\boldsymbol{h}} a^2 \frac{d\boldsymbol{h}}{d\boldsymbol{a}^1}\frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1} = [-(t-z)][\sigma'(a^2)]$$
$$= (w^2)^T \quad \begin{bmatrix} w_1^2 & w_2^2 & w_3^2 \end{bmatrix} \begin{bmatrix} \sigma'(a_1^1) & 0 & 0 \\ 0 & \sigma'(a_2^1) & 0 \\ 0 & 0 & \sigma'(a_3^1) \end{bmatrix} \frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1}$$

We can simplify this multiplication since the product of a vector with a diagonal matrix is equivalent to the element-wise multiplication (indicated with $\odot$) with the diagonal vector. Thus:

$$\frac{dJ}{d\boldsymbol{W}^1} = [-(t-z)][\sigma'(a^2)]\, [w_1^2 \quad w_2^2 \quad w_3^2] \odot [\sigma'(a_1^1) \quad \sigma'(a_2^1) \quad \sigma'(a_3^1)] \frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1} =$$
$$= \underbrace{[-(t-z)][\sigma'(a^2)]\, [w_1^2\sigma'(a_1^1) \quad w_2^2\sigma'(a_2^1) \quad w_3^2\sigma'(a_3^1)]}_{\mathbb{R}^{1\times 3}} \frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1}$$

Finally, we have only the last multiplication with the tensor $\frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1}$ to take care of.

Even though the mathematics of tensors might be hard, we just need to compute a product that resembles matrix multiplication.

Luckily, there is a very simple "trick" to compute our product, and it requires knowing only how to do matrix multiplications, so we have hope!

Example of Vector/Tensor multiplication

$$\boldsymbol{a} = [a_1 \quad a_2 \quad a_3],\ \mathbf{B} = \begin{bmatrix} \boldsymbol{B^1} \\ \boldsymbol{B^2} \end{bmatrix},\ B_i = \begin{bmatrix} b_{11}^i & b_{12}^i \\ b_{21}^i & b_{22}^i \\ b_{31}^i & b_{32}^i \end{bmatrix}$$

From the reasoning above, we know that for this example the shape we need is $\boldsymbol{a}\mathbf{B} \in \mathbb{R}^{2\times 2}$

Approach 1: compute the multiplications of $\boldsymbol{a}$ with the various $\boldsymbol{B}^i$ and then "collate" the results. In fact $\boldsymbol{a}\boldsymbol{B}^i \in \mathbb{R}^{1\times 2}$, so the shapes match.

In other words, $\boldsymbol{a}\mathbf{B} = \begin{bmatrix} \boldsymbol{a}\boldsymbol{B^1} \\ \boldsymbol{a}\boldsymbol{B^2} \end{bmatrix}$

Approach 2: we can obtain the same result first flattening the tensor in a matrix, and then re-shaping it to account for the "lost" dimension.

We define $\overline{\boldsymbol{B}} = \begin{bmatrix} b_{11}^1 & b_{12}^1 & b_{11}^2 & b_{12}^2 \\ b_{21}^1 & b_{22}^1 & b_{21}^2 & b_{22}^2 \\ b_{31}^1 & b_{31}^1 & b_{31}^2 & b_{32}^2 \end{bmatrix}$. We can now easily compute $\boldsymbol{a}\overline{\boldsymbol{B}} \in \mathbb{R}^{\mathbf{1}\times\mathbf{4}}$. We just have to reshape this vector back into our desired shape:

$\overline{\boldsymbol{aB}} = [(ab)_1 \quad (ab)_2 \quad (ab)_3 \quad (ab)_4]$, $\boldsymbol{a}\mathbf{B} = \begin{bmatrix} (ab)_1 & (ab)_2 \\ (ab)_3 & (ab)_4 \end{bmatrix}$

We are finally ready. As detailed before, $\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1} = \begin{bmatrix} \frac{\partial a_1^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_2^1}{\partial \boldsymbol{W}^1} \\ \frac{\partial a_3^1}{\partial \boldsymbol{W}^1} \end{bmatrix}$ is a tensor of shape $3 \times 3 \times 2$. Let's use the flattening approach, obtaining $\overline{\frac{\partial \boldsymbol{a}^1}{\partial \boldsymbol{W}^1}} \in \mathbb{R}^{3\times 6}$. We can compute the flattened gradient $\overline{\frac{\partial J}{\partial \boldsymbol{W}^1}} \in \mathbb{R}^{1\times 6}$ as:

$$\overline{\frac{\partial J}{\partial \boldsymbol{W}^1}} = [-(t-z)]\left[\sigma'^{(a^2)}\right] [w_1^2\sigma'(a_1^1) \quad w_2^2\sigma'(a_2^1) \quad w_3^2\sigma'(a_3^1)] \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{bmatrix} =$$

$$= [-(t-z)\sigma'(a^2)w_1^2\sigma'(a_1^1)x_1 \quad \dots \quad \dots \quad \dots \quad \dots \quad -(t-z)\sigma'(a^2)w_3^2\sigma'(a_3^1)x_2]$$

We can now reshape the matrix to obtain our gradient $\frac{\partial J}{\partial \boldsymbol{W}^1} \in \mathbb{R}^{3\times 2}$

$$\frac{\partial J}{\partial \boldsymbol{W}^1} = \begin{bmatrix} -(t-z)\sigma'(a^2)w_1^2\sigma'(a_1^1)x_1 & -(t-z)\sigma'(a^2)w_1^2\sigma'(a_1^1)x_2 \\ -(t-z)\sigma'(a^2)w_2^2\sigma'(a_2^1)x_1 & -(t-z)\sigma'(a^2)w_2^2\sigma'(a_2^1)x_2 \\ -(t-z)\sigma'(a^2)w_3^2\sigma'(a_3^1)x_1 & -(t-z)\sigma'(a^2)w_3^2\sigma'(a_3^1)x_2 \end{bmatrix}$$

Now that we know the result, we can notice that in our case, since the pre-activation a_i^1 only depends on the subset of weights connecting the input to that particular neuron, we have sparsity and regularities in $\frac{d\boldsymbol{a}^1}{d\boldsymbol{W}^1}$. We can avoid the computation with the tensor entirely, noticing that the result corresponds to a simpler outer product:

$$\frac{\partial J}{\partial \boldsymbol{W}^1} = \left([-(t-z)]\left[\sigma'^{(a^2)}\right] (\boldsymbol{w}^2)^T \odot [\sigma'(a_1^1) \quad \sigma'(a_2^1) \quad \sigma'(a_3^1)]\right)^T \boldsymbol{x}^T$$

Notice that we are multiplying the transpose of the first term that has shape 3×1 with the transpose of the input that has shape 1×2, thus obtaining a matrix of shape 3×2 that is the correct shape.

Extension to multiple examples in matrix notation

Neural Network implementations work directly with tensors, even when using multiple examples to compute the gradient (as in the first part of this lecture notes).
It is possible to exploit the matrix notation also in this case.

$$J = \frac{1}{N}[J^{(1)} \quad \dots \quad J^{(N)}] \begin{bmatrix} 1 \\ \dots \\ 1 \end{bmatrix}$$

Where $J^{(i)}$ is defined as detailed above for a single example.
In practice, multiple examples are managed with an added dimension to all the tensors discussed before. Still, in this case, the mathematics becomes more difficult due to the newly added dimension that would generate tensors with 4 dimensions.

From Gradient computation to Backpropagation

Let's consider the backpropagation algorithm, in particular the backward pass. We will not consider the regularization terms for this exercise.

Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses in addition to the input $\boldsymbol{x}$ a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \dots, 1$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g}$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top}$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for

Exercise: show that backpropagation correctly computes the gradients for the 2-layer network considered before.

Solution:
In our notation, we referred to $\hat{y}$ as z.
We have 2 layers in total (1 hidden layer and 1 output layer). Thus, l=2.
Before the for loop, we initialize $g \leftarrow \nabla_z J$
First iteration of for loop: k=2
We update g with the element-wise multiplication with $f'(a^2)$. In our case $f = \sigma$ and $a^2 \in \mathbb{R}$, thus the element-wise product corresponds to the classical product that we have in our derivation.
Now we have $g \leftarrow \nabla_z J \sigma'(a^2)$
This is exactly the update for the bias (we didn't compute it before, it was left as exercise).
The algorithm now computes $\nabla_{\boldsymbol{W}^2} J = \frac{\partial J}{\partial \boldsymbol{W}^2} = g\ (\boldsymbol{h}^1)^T$
In our notation, we referred to $\boldsymbol{h}^1$ simply as $\boldsymbol{h}$.
Thus, expanding the equation, the backprop algorithm computes

$$\nabla_{\boldsymbol{W}^2} J = \frac{\partial J}{\partial \boldsymbol{W}^2} = g\ (\boldsymbol{h})^T = \nabla_z J \sigma'(a^2)(\boldsymbol{h})^T$$

That is exactly the gradient we computed in matrix notation.
Before finishing the for loop, the algorithm updates g multiplying it by the transpose of the weights of the current layer. Notice that, according to our computation, this is the third term of $\frac{dJ}{d\boldsymbol{W}^1}$. Notice also that the backpropagation algorithm as presented in the book uses another convention for gradients and Jacobian matrices, that makes more complicated the operations (in fact, the multiplication with $\left(\boldsymbol{W}^{(k)}\right)^T$ is on the left, while our notation makes everything simpler and we can just compute the multiplications in order).
Thus, g is updated as $g \leftarrow g\left(\boldsymbol{W}^{(k)}\right)^T$. In our case $\boldsymbol{w}^2$ is a vector, and with the aforementioned consideration we have that:

$$g \leftarrow g \cdot (\boldsymbol{w}^2)^T = \nabla_z J \sigma'(a^2)(\boldsymbol{w}^2)^T$$

Second iteration of for loop: k=2
Similarly as before, we update g with the element-wise multiplication with $f'(\boldsymbol{a}^1)$, that in our case is the sigmoid function:

$$g \leftarrow g \odot f'(\boldsymbol{a}^1) = \nabla_z J \sigma'(a^2)(\boldsymbol{w}^2)^T \odot \sigma'(\boldsymbol{a}^1)$$

Notice that this term is exactly the fourth term in our derivation of $\frac{dJ}{d\boldsymbol{W}^1}$.
Again, similarly as before, this term is the correct update for the bias, while to compute $\nabla_{\boldsymbol{W}^1} J$ we have to multiply g with the transpose of $\boldsymbol{h}^{k-1} = \boldsymbol{h}^{1-1} = \boldsymbol{x}$.
Again for the different notations, the multiplication with $\boldsymbol{x}^T$ corresponds to our "trick" of using the outer product. This was the last missing term of $\frac{dJ}{d\boldsymbol{W}^1}$, that is thus computed correctly.

Notice how the variable g allows not to re-compute the first terms of the gradients (this is where the computational efficiency arises).
However, we are required to store both $\boldsymbol{h}$ and $\boldsymbol{a}$ for all the layers (see forward part of backpropagation)