Exercises on Neural Networks for Natural Language Processing

Ion Androutsopoulos, 2017–18

Submit as a group of 3–4 members a report for exercise 10 (max. 5 pages, PDF format). Include in your report all the required information, especially experimental results. Do not include code in the report, but include a link to a shared folder or repository (e.g. in Dropbox, GitHub, Bitbucket) containing your code.

1. Show that a Perceptron (single neuron) with (i) a sign activation function or (ii) a sigmoid activation function is a linear separator. (iii) Show that a two-level network of Perceptrons (like the one we used to implement the XOR gate) can learn any logical function. Hint: at the first level, use an AND gate for each row of the logical function's truth table; at the second level, use an OR gate. (iv) Consider a training dataset for binary classification (classes: true, false) and Boolean features. Explain why a two-level Perceptron network, like the one of the previous question, with one first-level gate per training instance can learn the training dataset perfectly, but may not perform well on fresh test data.

2. (i) We wish to train a (single) Perceptron to separate the instances of the two classes (black and white dots, inside and outside of a circle) of the figure on the right. There are only two (real-valued) features, corresponding to the two axes. Explain why the Perceptron cannot learn to correctly separate the two classes using the current two features.

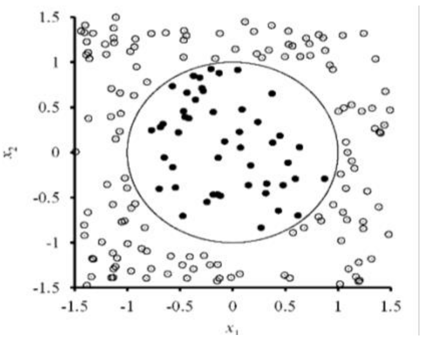


Image from the book of Russel & Norvig; see references in the slides.

Answer: The Perceptron is a linear classifier, i.e., it learns a point (for one feature), a straight line (for two features), a plane (for three features), or more generally a hyper-plane (for more features), and classifies unseen instances by examining if they fall above or below the hyper-plane. The dataset of the figure is not linearly separable with the current two features, i.e., there is no straight line that separates the black from the white dots. Hence, a single Perceptron cannot learn to separate the two classes with the current features.

(ii) Propose a mapping from the feature vector of each instance to a single real number (a single real-valued feature), so that the new (single) feature will allow the Perceptron to correctly separate the two classes.

Answer: We can represent each instance by its distance from the center of the circle. Then all the instances will be along the axis of the new, single feature (distance from the center), the black dots will be on the left of the value that corresponds to the radius (approximately 1), and the white dots will be on the right of the radius value. With the new (single feature) representation, the classes are linearly separable, hence the Perceptron can learn to correctly separate them.

3. (i) Two students are discussing how the Perceptron (single neuron) relates to a logistic regression classifier. The first student claims that a (binary) logistic regression classifier is the same as a (single neuron) Perceptron with a sigmoid activation function. To support her view, she wrote down the formulae that compute the output of the Perceptron and the probability that the logistic regression classifier assigns to the positive class, in both cases given an input vector $\vec{x}$. Write down the formulae. What do they show?

Answer: The output of the Perceptron with a sigmoid activation function is:

$$\Phi(\sum w_l x_l) = \Phi(\vec{w} \cdot \vec{x}) = 1/(1+e^{-\vec{w}\cdot\vec{x}})$$

The probability that the logistic regression classifier assigns to the positive class is:

$$P(c_+ \mid \vec{x}) = 1/(1+e^{-\vec{w}\cdot\vec{x}})$$

The formulae show that if we use the same weights $\vec{w}$, the output of the Perceptron will be the same as the probability of the positive class of logistic regression, which seems to agree with the claim of the first student.

(ii) The second student, however, responded that the Perceptron and logistic regression learn different weights, even if they use the same training dataset, the same initial weights, and the same optimizer. To support her claim, she wrote down the weight update rules of the Perceptron (with sigmoid activation function, slide 15) and logistic regression (with stochastic gradient ascent). Write the update rules. What do they show?

Answer: The weight update rule of the Perceptron (with sigmoid activation function) is:

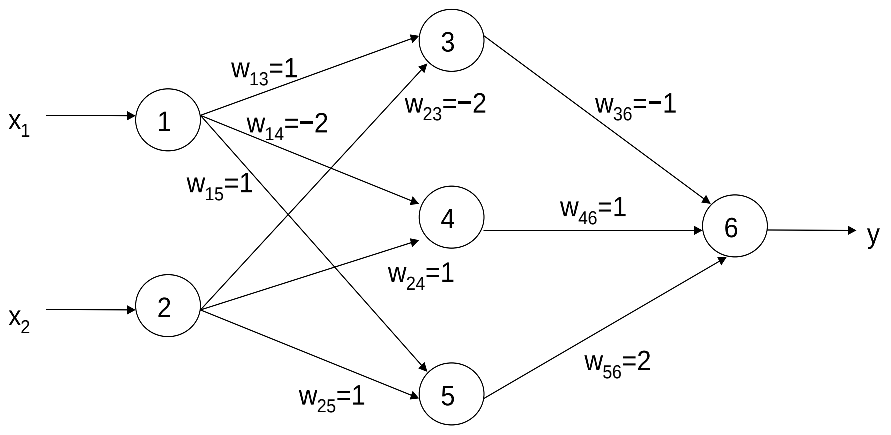
$$w_l \leftarrow w_l + \eta \cdot \Phi(\vec{w}\cdot\vec{x}^{(i)}) \cdot (1-\Phi(\vec{w}\cdot\vec{x}^{(i)})) \cdot (t^{(i)} - \Phi(\vec{w}\cdot\vec{x}^{(i)})) \cdot x_l^{(i)}$$

The weight update rule of logistic regression (with stochastic gradient ascent, without regularization) is:

$$w_l \leftarrow w_l + \eta \cdot [t^{(i)} - P(c_+ \mid \vec{x}^{(i)})] \cdot x_l^{(i)}$$

The update rules are indeed different. This is because the Perceptron that we considered in the slides tries to minimize the squared error loss, whereas logistic regression tries to minimize the cross-entropy (or to maximize the conditional log-likelihood) of the training data. Hence, the second student is right, that in general the Perceptron will learn different weights than logistic regression. However, if we used the cross-entropy loss in the Perceptron too (and the same regularization and optimizer), we would come up with the same update rules, which would agree with the first student's opinion.

4. In the following network, all the neurons use a sign activation function with a threshold of 0, i.e., $\Phi(x) = 1$ if $x \geq 0$, $\Phi(x) = -1$ if $x < 0$. The current weights have the values shown. There are no weights on the inputs of neurons 1 and 2. (i) We feed the network with the training instance $x_1 = 1$, $x_2 = 1$, for which the correct output is $t = -1$. What is the output y of the network? (ii) We use backpropagation, with update rule $w_{ij} \leftarrow w_{ij} + 0.1 \cdot x_{ij} \cdot \delta_j$. Assume that for the output neuron $\delta_6 = t - y$ and that for each hidden layer neuron $\delta_j = \sum w_{jk} \cdot \delta_k$. Compute the new weights.



Answer: (i) The output of the network is:

$$y = \Phi(w_{36}x_{36} + w_{46}x_{46} + w_{56}x_{56})$$

$$= \Phi(w_{36}\Phi(w_{13}x_{13} + w_{23}x_{23}) + w_{46}\Phi(w_{14}x_{14} + w_{24}x_{24}) + w_{56}\Phi(w_{15}x_{15} + w_{25}x_{25}))$$

$$= \Phi(w_{36}\Phi\big(w_{13}\Phi(x_1) + w_{23}\Phi(x_2)\big) + w_{46}\Phi\big(w_{14}\Phi(x_1) + w_{24}\Phi(x_2)\big) + w_{56}\Phi\big(w_{15}\Phi(x_1) + w_{25}\Phi(x_2)\big))$$

$$= \Phi(-1\cdot\Phi(1\cdot 1 + (-2)\cdot 1) + 1\cdot\Phi\big((-2)\cdot 1 + 1\cdot 1\big) + 2\cdot\Phi(1\cdot 1 + 1\cdot 1))$$

$$= \Phi\big(-\Phi(-1) + \Phi(-1) + 2\cdot\Phi(2)\big)$$

$$= \Phi(1 - 1 + 2) = \Phi(2) = 1$$

(ii) We first compute δ_j $(j = 3, \dots, 6)$:

$$\begin{aligned}
\delta_6 &= t - y = -1 - 1 = -2\\
\delta_3 &= w_{36}\cdot\delta_6 = (-1)\cdot(-2) = 2\\
\delta_4 &= w_{46}\cdot\delta_6 = 1\cdot(-2) = -2\\
\delta_5 &= w_{56}\cdot\delta_6 = 2\cdot(-2) = -4
\end{aligned}$$

We then compute the new weights:

$$\begin{aligned}
w_{13} &\longleftarrow w_{13} + 0.1\cdot x_{13}\cdot\delta_3 = 1 + 0.1\cdot 1\cdot 2 = 1.2\\
w_{14} &\longleftarrow w_{14} + 0.1\cdot x_{14}\cdot\delta_4 = -2 + 0.1\cdot 1\cdot(-2) = -2.2\\
w_{15} &\longleftarrow w_{15} + 0.1\cdot x_{15}\cdot\delta_5 = 1 + 0.1\cdot 1\cdot(-4) = 0.6\\
w_{23} &\longleftarrow w_{23} + 0.1\cdot x_{23}\cdot\delta_3 = -2 + 0.1\cdot 1\cdot 2 = -1.8\\
w_{24} &\longleftarrow w_{24} + 0.1\cdot x_{24}\cdot\delta_4 = 1 + 0.1\cdot 1\cdot(-2) = 0.8\\
w_{25} &\longleftarrow w_{25} + 0.1\cdot x_{25}\cdot\delta_5 = 1 + 0.1\cdot 1\cdot(-4) = 0.6\\
w_{36} &\longleftarrow w_{36} + 0.1\cdot x_{36}\cdot\delta_6 = -1 + 0.1\cdot(-1)\cdot(-2) = -0.8\\
w_{46} &\longleftarrow w_{46} + 0.1\cdot x_{46}\cdot\delta_6 = 1 + 0.1\cdot(-1)\cdot(-2) = 1.2\\
w_{56} &\longleftarrow w_{56} + 0.1\cdot x_{56}\cdot\delta_6 = 2 + 0.1\cdot 1\cdot(-2) = 1.8
\end{aligned}$$

5. Show that without activation functions, a multi-layer neural network is equivalent to applying a linear transformation to the input, i.e., the output can be written as $\vec{o} = W\vec{x} + b$, where W is a weights matrix, $b \in \mathbb{R}$ is a bias term, and $\vec{x}\epsilon\mathbb{R}^n$ is the input feature vector.

6. Derive the weight update rules of slide 21 (backpropagation for the MLP of slides 18–19).

Answer (based on section 4.5.2 of the book «Machine Learning» by T. Mitchell, 1997, McGraw Hill – see Exercises 7 and 8 below for an alternative solution based on the computation graph of the neural network):

Let W be a single vector containing all the weights $w_{i,j}$ (of all the layers) of the network, where $w_{i,j}$ denotes the weight of the connection from neuron i to neuron j. Each time we consider a training example, we compute the squared error loss for this particular example:

$$E(W) = \frac{1}{2}\sum_{k\in Outputs}(t_k - o_k)^2$$

where $Outputs$ is the set containing the output layer neurons (more precisely, the numbers that we use to refer to the neurons of the output layer), o_k is the output of neuron k, and t_k is the correct output for neuron k. We update W (all the weights together) by taking a step towards $-\nabla E(W)$:

$$W \leftarrow W - \eta \nabla E(W)$$

which means that we update the weights as follows:

$$\langle \dots, w_{1,4}, \dots, w_{i,j}, \dots, w_{4,8}, \dots \rangle \leftarrow \langle \dots, w_{1,4}, \dots, w_{i,j}, \dots, w_{4,8}, \dots \rangle$$
$$-\eta \left\langle \dots, \frac{\partial E(W)}{\partial w_{1,4}}, \dots, \frac{\partial E(W)}{\partial w_{i,j}}, \dots, \frac{\partial E(W)}{\partial w_{4,8}}, \dots \right\rangle$$

where η is (in the simplest case) a small positive constant (e.g., 0.1).

Each weight $w_{i,j}$ inside W is modified as follows:

$$w_{i,j} \leftarrow w_{i,j} - \eta \frac{\partial E(W)}{\partial w_{i,j}}$$

Let $s_j = \sum_{i'} w_{i',j} x_{i',j} = \sum_{i'} w_{i',j} o_{i'}$, where i' ranges over the neurons that provide input to neuron j and $o_{i'}$ is the output of neuron i', i.e., s_j is the weighted sum of the inputs of neuron j, before applying the activation function. The weight $w_{i,j}$ affects the outputs of the neurons of the output layer, hence also the loss $E(W)$, only through s_j (only by affecting s_j). Therefore, using the chain rule of derivatives, we obtain:[1]

$$\frac{\partial E(W)}{\partial w_{i,j}} = \frac{\partial E(W)}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}}$$

Hence:

$$w_{i,j} \leftarrow w_{i,j} - \eta \frac{\partial E(W)}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}} =$$
$$w_{i,j} - \eta \frac{\partial E(W)}{\partial s_j} \frac{\partial \sum_{i'} w_{i',j} x_{i',j}}{\partial w_{i,j}} = w_{i,j} - \eta \frac{\partial E(W)}{\partial s_j} x_{i,j}$$

***Case 1: If neuron j is in the output layer,** then s_j affects the outputs of the neurons of the output layer (actually, only the output of neuron j), hence also the loss $E(W)$, only through $o_j = \Phi(s_j)$, where Φ is the activation function of neuron j. Therefore, using the chain rule, we obtain:*

$$\frac{\partial E(W)}{\partial s_j} = \frac{\partial E(W)}{\partial o_j} \frac{\partial o_j}{\partial s_j}$$

Assuming that the activation function of neuron j is the sigmoid, $\Phi(s_j) = \sigma(s_j)$, we obtain:

$$\frac{\partial E(W)}{\partial s_j} = \frac{\partial}{\partial o_j} \left(\frac{1}{2} \sum_{k \in Outputs} (t_k - o_k)^2 \right) \frac{\partial \sigma(s_j)}{\partial s_j} =$$

[1] Βλ. https://en.wikipedia.org/wiki/Chain_rule

$$\left(\sum_{k \in Outputs} \frac{\partial}{\partial o_j}\left(\frac{1}{2}(t_k - o_k)^2\right)\right)\sigma(s_j)\left(1-\sigma(s_j)\right) =$$

$$\left(\sum_{k \in Outputs} (t_k - o_k)\frac{\partial (t_k - o_k)}{\partial o_j}\right) o_j(1-o_j) =$$
$$(t_j - o_j)(-1)o_j(1-o_j) = -(t_j - o_j)o_j(1-o_j)$$

Therefore, in this case (when neuron j is in the output layer), the update rule is:

$$w_{i,j} \leftarrow w_{i,j} + \eta\left((t_j - o_j)o_j(1-o_j)\right)x_{i,j}$$

By setting $\delta_j = -\frac{\partial E(W)}{\partial s_j}$, the update rule of this case becomes:

$$w_{i,j} \leftarrow w_{i,j} + \eta \delta_j x_{i,j}$$

where:

$$\delta_j = (t_j - o_j)o_j(1-o_j)$$

***Case 2: If neuron j is in the hidden layer, then** s_j affects the outputs of the neurons of the output layer, hence also the loss $E(W)$, only through the s_k of each neuron $k \in Downstream(j)$ to which neuron j provides (directly) input.*[2] *Therefore, using the chain rule and setting again $\delta_k = -\frac{\partial E(W)}{\partial s_k}$, we obtain:*

$$\frac{\partial E(W)}{\partial s_j} = \sum_{k \in Downstream(j)} \frac{\partial E(W)}{\partial s_k}\frac{\partial s_k}{\partial s_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial s_k}{\partial s_j}$$

s_j affects each s_k only through o_j. Therefore, using the chain rule and assuming again that neuron j has a sigmoid activation function, i.e., $o_j = \sigma(s_j)$, we obtain:

$$\frac{\partial E(W)}{\partial s_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial s_k}{\partial o_j}\frac{\partial o_j}{\partial s_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial \sum_{j'} w_{j',k} o_{j'}}{\partial o_j}\frac{\partial o_j}{\partial s_j} =$$

$$\sum_{k \in Downstream(j)} -\delta_k w_{j,k}\frac{\partial \sigma(s_j)}{\partial s_j} = \sum_{k \in Downstream(j)} -\delta_k w_{j,k}\,\sigma(s_j)\left(1-\sigma(s_j)\right) =$$

$$\sum_{k \in Downstream(j)} -\delta_k w_{j,k}\, o_j(1-o_j) = -o_j(1-o_j)\sum_{k \in Downstream(j)} \delta_k w_{j,k}$$

Therefore, in this case (when neuron j is in the hidden layer), the update rule is:

[2] In the network of slide 18, if neuron j is in the single hidden layer and every neuron of the hidden layer is connected to every neuron of the output layer, then $Downstream(j) = Outputs$. The notation and update rules of this exercise, however, can also be used when there are multiple hidden layers.

$$w_{i,j} \leftarrow w_{i,j} + \eta \left(o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{j,k} \right) x_{i,j}$$

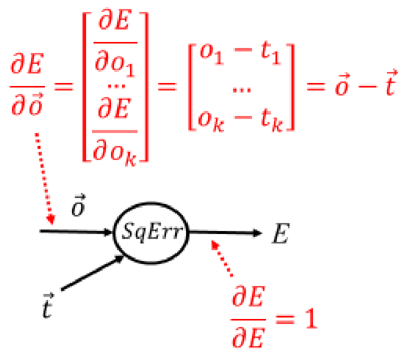
Since $\delta_j = -\frac{\partial E_d(W)}{\partial net_j}$, *we can write again the update rule as:*

$$w_{i,j} \leftarrow w_{i,j} + \eta \delta_j x_{i,j}$$

but in this case:

$$\delta_j = \left(o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{j,k} \right)$$

7. Confirm the computation of $\frac{\partial E}{\partial \vec{o}}$ in the computation graph of slide 32.



$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = \begin{bmatrix} o_1 - t_1 \\ \cdots \\ o_k - t_k \end{bmatrix} = \vec{o} - \vec{t}$$

Answer: The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_i} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial o_i}$ *(a single element of the gradient):*

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i} \sum_{j=1}^{k} \frac{1}{2} \left(t_j - o_j\right)^2 = \frac{\partial}{\partial o_i} \frac{1}{2} (t_i - o_i)^2 = \frac{1}{2} \cdot 2 \cdot (t_i - o_i) \cdot \frac{\partial}{\partial o_i} (t_i - o_i)$$
$$= (t_i - o_i) \cdot (-1) = (o_i - t_i)$$

Hence:

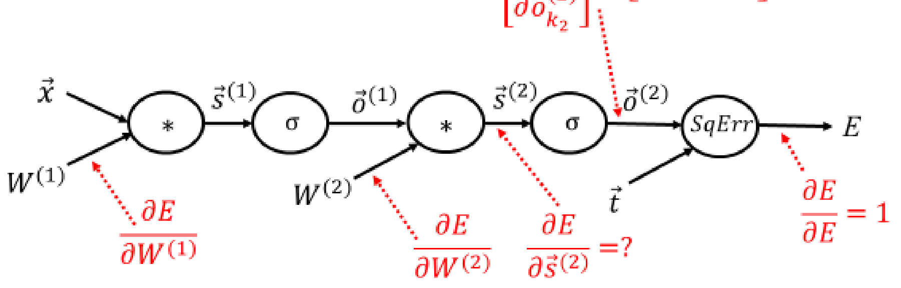
$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_i} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = \begin{bmatrix} o_1 - t_1 \\ \cdots \\ o_i - t_i \\ \cdots \\ o_k - t_k \end{bmatrix} = \vec{o} - \vec{t}$$

Note: We do not need to compute $\frac{\partial E}{\partial \vec{t}}$, because we do not update $\vec{t}$ (the correct prediction).

8. (i) Draw the computation graph of the neural network of slides 18–19 (the one used in exercise 6 above) and compute the gradient $\frac{\partial E}{\partial \vec{o}^{(2)}}$.

Answer: The graph is the same as in slide 32, but with an extra sigmoid before the squared error loss. (We would use the extra sigmoid if we wanted each output of the network to predict the probability that the input text belongs in the corresponding class, without the classes being mutually exclusive; for mutually exclusive classes, we would use a softmax instead of the extra sigmoid.) The gradient $\frac{\partial E}{\partial \vec{o}^{(2)}}$ was computed as in Exercise 7.

$$\frac{\partial E}{\partial \vec{o}^{(2)}} = \begin{bmatrix} \frac{\partial E}{\partial o_1^{(2)}} \\ \cdots \\ \frac{\partial E}{\partial o_{k_2}^{(2)}} \end{bmatrix} = \begin{bmatrix} o_1^{(2)} - t_1 \\ \cdots \\ o_{k_2}^{(2)} - t_{k_2} \end{bmatrix} = \vec{o}^{(2)} - \vec{t}$$


$\vec{x}$
$W^{(1)}$
$*$
$\vec{s}^{(1)}$
σ
$\vec{o}^{(1)}$
$W^{(2)}$
$*$
$\vec{s}^{(2)}$
σ
$\vec{o}^{(2)}$
$SqErr$
$\vec{t}$
E
$\frac{\partial E}{\partial W^{(1)}}$
$\frac{\partial E}{\partial W^{(2)}}$
$\frac{\partial E}{\partial \vec{s}^{(2)}} =?$
$\frac{\partial E}{\partial E} = 1$

(ii) Show that for a sigmoid node $\sigma(\vec{s}) = \vec{o}$, $\frac{\partial E}{\partial \vec{s}}$ can be computed as follows, where J is the Jacobian matrix.[3]

[3] See https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant.

$$\begin{tikzcd}
\vec{s}\in\mathbb{R}^k \arrow[r] & \sigma \arrow[r] & \vec{o}\in\mathbb{R}^k \arrow[l, dotted, "\frac{\partial E}{\partial \vec{o}}\in\mathbb{R}^k"]
\end{tikzcd}$$

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & \frac{\partial \sigma(s_2)}{\partial s_1} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_1} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_i} & \frac{\partial \sigma(s_2)}{\partial s_i} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_k} & \frac{\partial \sigma(s_2)}{\partial s_k} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = J^T \frac{\partial E}{\partial \vec{o}}$$

$$= \begin{bmatrix} \sigma(s_1)(1-\sigma(s_1)) & 0 & \cdots & 0 \\ 0 & \sigma(s_2)(1-\sigma(s_2)) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma(s_k)(1-\sigma(s_k)) \end{bmatrix} \frac{\partial E}{\partial \vec{o}}$$

Answer: The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial s_i}$ *(a single element of the gradient). By the chain rule of derivatives, we obtain:*

$$\frac{\partial E}{\partial s_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial s_i}$$

However, each s_i *affects only* $o_i = \sigma(s_i)$. *It does not affect any other* $o_j = \sigma(s_j)$, *for* $j \neq i$. *Hence,* $\frac{\partial o_j}{\partial s_i} = 0$ *for* $j \neq i$, *and we obtain:*

$$\frac{\partial E}{\partial s_i} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial s_i} = \frac{\partial E}{\partial o_i} \frac{\partial \sigma(s_i)}{\partial s_i} = \frac{\partial E}{\partial o_i} \sigma(s_i)(1-\sigma(s_i))$$

where we have use the property of the sigmoid that $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$.

Therefore:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial o_1}\frac{\partial \sigma(s_1)}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial o_i}\frac{\partial \sigma(s_i)}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial o_k}\frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial o_1}\sigma(s_1)\big(1-\sigma(s_1)\big) \\ \vdots \\ \frac{\partial E}{\partial o_i}\sigma(s_i)\big(1-\sigma(s_i)\big) \\ \vdots \\ \frac{\partial E}{\partial o_k}\sigma(s_k)\big(1-\sigma(s_k)\big) \end{bmatrix}$$

The latter can also be written as:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & 0 & \dots & 0 \\ 0 & \frac{\partial \sigma(s_2)}{\partial s_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} =$$

$$= \begin{bmatrix} \sigma(s_1)\big(1-\sigma(s_1)\big) & 0 & \dots & 0 \\ 0 & \sigma(s_2)\big(1-\sigma(s_2)\big) & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma(s_k)\big(1-\sigma(s_k)\big) \end{bmatrix} \frac{\partial E}{\partial \vec{o}}$$

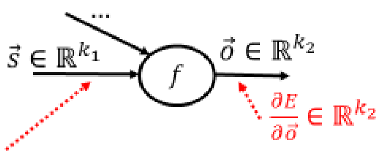
More generally, it can be written as:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & \frac{\partial \sigma(s_2)}{\partial s_1} & \dots & \frac{\partial \sigma(s_k)}{\partial s_1} \\ \frac{\partial \sigma(s_1)}{\partial s_2} & \frac{\partial \sigma(s_2)}{\partial s_2} & \dots & \frac{\partial \sigma(s_k)}{\partial s_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_k} & \frac{\partial \sigma(s_2)}{\partial s_k} & \dots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = J^T \frac{\partial E}{\partial \vec{o}}$$

Where J is the Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & \frac{\partial \sigma(s_1)}{\partial s_2} & \dots & \frac{\partial \sigma(s_1)}{\partial s_k} \\ \frac{\partial \sigma(s_2)}{\partial s_1} & \frac{\partial \sigma(s_2)}{\partial s_2} & \dots & \frac{\partial \sigma(s_2)}{\partial s_k} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_k)}{\partial s_1} & \frac{\partial \sigma(s_k)}{\partial s_2} & \dots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix}$$

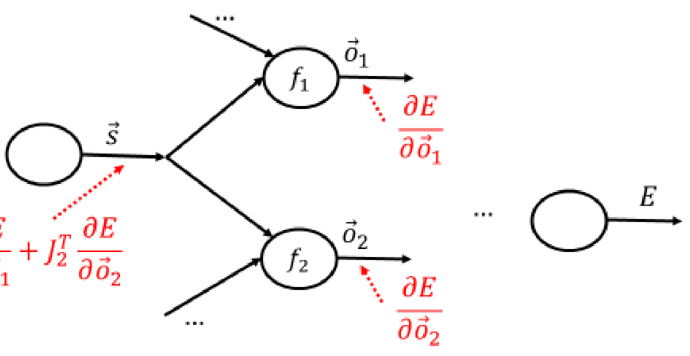
The latter applies more generally. For a node that computes $f(\vec{s}, \dots) = \vec{o}$, we can compute $\frac{\partial E}{\partial \vec{s}}$ as follows (provided that $\vec{s}$ is fed only to the f node):


...
$\vec{s}\in\mathbb{R}^{k_1}$
f
$\vec{o}\in\mathbb{R}^{k_2}$
$\frac{\partial E}{\partial \vec{o}}\in\mathbb{R}^{k_2}$

$$\frac{\partial E}{\partial \vec{s}}=\begin{bmatrix}\frac{\partial E}{\partial s_1}\\ \vdots\\ \frac{\partial E}{\partial s_i}\\ \vdots\\ \frac{\partial E}{\partial s_{k_1}}\end{bmatrix}=\begin{bmatrix}\frac{\partial o_1}{\partial s_1} & \frac{\partial o_2}{\partial s_1} & \cdots & \frac{\partial o_{k_2}}{\partial s_1}\\ \vdots & \vdots & \vdots & \vdots\\ \frac{\partial o_1}{\partial s_i} & \frac{\partial o_2}{\partial s_i} & \cdots & \frac{\partial o_{k_2}}{\partial s_i}\\ \vdots & \vdots & \vdots & \vdots\\ \frac{\partial o_1}{\partial s_{k_1}} & \frac{\partial o_2}{\partial s_{k_1}} & \cdots & \frac{\partial o_{k_2}}{\partial s_{k_1}}\end{bmatrix}\begin{bmatrix}\frac{\partial E}{\partial o_1}\\ \vdots\\ \frac{\partial E}{\partial o_i}\\ \vdots\\ \frac{\partial E}{\partial o_{k_2}}\end{bmatrix}=J^T\frac{\partial E}{\partial \vec{o}}$$

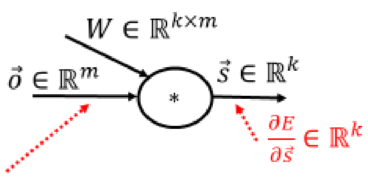
(Check that this is also true for $\frac{\partial E}{\partial \vec{o}}$ in exercise 7.)

If $\vec{s}$ is fed to two (or more) nodes f_1, f_2, we have to add the gradients for $\frac{\partial E}{\partial \vec{s}}$ that we get from f_1, f_2:


...
f_1
$\vec{o}_1$
$\frac{\partial E}{\partial \vec{o}_1}$
$\vec{s}$
...
E
f_2
$\vec{o}_2$
$\frac{\partial E}{\partial \vec{o}_2}$
...

$$\frac{\partial E}{\partial \vec{s}}=J_1^T\frac{\partial E}{\partial \vec{o}_1}+J_2^T\frac{\partial E}{\partial \vec{o}_2}$$

(iii) Show that for a matrix-vector multiplication node $W\vec{o}=\vec{s}$, $\frac{\partial E}{\partial \vec{o}}$ can be computed as follows:


$W\in\mathbb{R}^{k\times m}$
$\vec{o}\in\mathbb{R}^{m}$
$*$
$\vec{s}\in\mathbb{R}^{k}$
$\frac{\partial E}{\partial \vec{s}}\in\mathbb{R}^{k}$

$$\frac{\partial E}{\partial \vec{o}}=J_{\vec{o}}^T\frac{\partial E}{\partial \vec{s}}=W^T\frac{\partial E}{\partial \vec{s}}$$

Answer:

$$\vec{s}=\begin{bmatrix}s_1\\ s_2\\ s_3\\ \cdots\\ s_k\end{bmatrix}=W\vec{o}=\begin{bmatrix}w_{1,1} & w_{2,1} & \cdots & w_{m,1}\\ w_{1,2} & w_{2,2} & \cdots & w_{m,2}\\ w_{1,3} & w_{2,3} & \cdots & w_{m,3}\\ \cdots & \cdots & \cdots & \cdots\\ w_{1,k} & w_{2,k} & \cdots & w_{m,k}\end{bmatrix}\begin{bmatrix}o_1\\ o_2\\ o_3\\ \cdots\\ o_m\end{bmatrix}=\begin{bmatrix}w_{1,1}o_1+w_{2,1}o_2+\cdots+w_{m,1}o_m\\ w_{1,2}o_1+w_{2,2}o_2+\cdots+w_{m,2}o_m\\ w_{1,3}o_1+w_{2,3}o_2+\cdots+w_{m,3}o_m\\ \cdots\\ w_{1,k}o_1+w_{2,k}o_2+\cdots+w_{m,k}o_m\end{bmatrix}$$

The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_m} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial o_i}$ *(a single element of the gradient). By the chain rule of derivatives, we obtain:*

$$\frac{\partial E}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial o_i}$$

According to the equations for $\vec{s} = W\vec{o}$ *above:*

$$s_j = w_{1,j}o_1 + w_{2,j}o_2 + \cdots + w_{i,j}o_i + \cdots + w_{m,j}o_m$$

Hence:

$$\frac{\partial s_j}{\partial o_i} = w_{i,j}$$

Therefore:

$$\frac{\partial E}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} w_{i,j}$$

which can also be written as:

$$\frac{\partial E}{\partial o_i} = \begin{bmatrix} \frac{\partial s_1}{\partial o_i} & \frac{\partial s_2}{\partial o_i} & \cdots & \frac{\partial s_k}{\partial o_i} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} w_{i,1} & w_{i,2} & \cdots & w_{i,k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix}$$

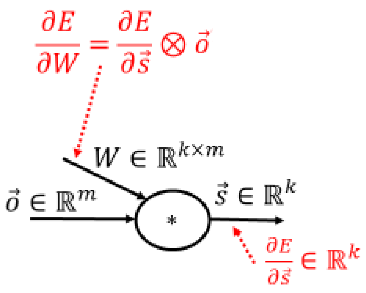
Hence, for the overall gradient:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial s_1}{\partial o_1} & \frac{\partial s_2}{\partial o_1} & \cdots & \frac{\partial s_k}{\partial o_1} \\ \frac{\partial s_1}{\partial o_2} & \frac{\partial s_2}{\partial o_2} & \cdots & \frac{\partial s_k}{\partial o_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial s_1}{\partial o_i} & \frac{\partial s_2}{\partial o_i} & \cdots & \frac{\partial s_k}{\partial o_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial s_1}{\partial o_m} & \frac{\partial s_2}{\partial o_m} & \cdots & \frac{\partial s_k}{\partial o_m} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w_{i,1} & w_{i,2} & \cdots & w_{i,k} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} =$$

$$J_{\vec{o}}^T \frac{\partial E}{\partial \vec{s}} = W^T \frac{\partial E}{\partial \vec{s}}$$

Note: We prefer to use matrix operators, which can be efficiently computed using highly optimized algorithms and GPUs, rather than relying on our own for-loops (e.g., in our own Python scripts) to compute individual elements of matrices, which is much slower.

(iv) Show that for a matrix-vector multiplication node $W\vec{o} = \vec{s}$, $\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$, where $\otimes$ denotes the outer product.[4]



$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$
$W \in \mathbb{R}^{k\times m}$
$\vec{o} \in \mathbb{R}^m$
*
$\vec{s} \in \mathbb{R}^k$
$\frac{\partial E}{\partial \vec{s}} \in \mathbb{R}^k$

Answer: Recall that we use the following notation for the elements of W:

$$W = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{m,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{m,2} \\ w_{1,3} & w_{2,3} & \cdots & w_{m,3} \\ \cdots & \cdots & \cdots & \cdots \\ w_{1,k} & w_{2,k} & \cdots & w_{m,k} \end{bmatrix}$$

The gradient that we need to compute is:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,1}} \\ \frac{\partial E}{\partial w_{1,2}} & \frac{\partial E}{\partial w_{2,2}} & \cdots & \frac{\partial E}{\partial w_{m,2}} \\ \frac{\partial E}{\partial w_{1,3}} & \frac{\partial E}{\partial w_{2,3}} & \cdots & \frac{\partial E}{\partial w_{m,3}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial E}{\partial w_{1,k}} & \frac{\partial E}{\partial w_{2,k}} & \cdots & \frac{\partial E}{\partial w_{m,k}} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial w_{i,j}}$ (a single element of the gradient). By the chain rule of derivatives, we obtain:

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{l=1}^{k} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial w_{i,j}}$$

According to the equations for $\vec{s} = W\vec{o}$ in part (iii) of the exercise:

$$s_l = w_{1,l}o_1 + w_{2,l}o_2 + \cdots + w_{i,l}o_i + \cdots + w_{m,l}o_m$$

[4] See https://en.wikipedia.org/wiki/Matrix_multiplication#Outer_product.

Hence:

$$\frac{\partial s_l}{\partial w_{i,j}} = 0, \text{for } l \neq j$$

and:

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{l=1}^{k} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}}$$

Given that:

$$s_j = w_{1,j} o_1 + w_{2,j} o_2 + \cdots + w_{i,j} o_i + \cdots + w_{m,j} o_m$$

we obtain:

$$\frac{\partial s_j}{\partial w_{i,j}} = o_i$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} o_i$$

Going back to the overall gradient:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,1}} \\ \frac{\partial E}{\partial w_{1,2}} & \frac{\partial E}{\partial w_{2,2}} & \cdots & \frac{\partial E}{\partial w_{m,2}} \\ \frac{\partial E}{\partial w_{1,3}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,3}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial w_{1,k}} & \frac{\partial E}{\partial w_{2,k}} & \cdots & \frac{\partial E}{\partial w_{m,k}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial s_1} o_1 & \frac{\partial E}{\partial s_1} o_2 & \cdots & \frac{\partial E}{\partial s_1} o_m \\ \frac{\partial E}{\partial s_2} o_1 & \frac{\partial E}{\partial s_2} o_2 & \cdots & \frac{\partial E}{\partial s_2} o_m \\ \frac{\partial E}{\partial s_3} o_1 & \frac{\partial E}{\partial s_3} o_2 & \cdots & \frac{\partial E}{\partial s_3} o_m \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial s_k} o_1 & \frac{\partial E}{\partial s_k} o_2 & \cdots & \frac{\partial E}{\partial w_{s_k}} o_m \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \frac{\partial E}{\partial s_3} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} \begin{bmatrix} o_1 & o_2 & \cdots & o_m \end{bmatrix} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$$

(v) Use the conclusions of parts (i)–(iv) of this exercise, to derive the backpropagation update rules for $W^{(1)}$ and $W^{(2)}$. Confirm that they are the same as the ones derived in Exercise 6 (also shown on slide 21).

Answer: For the weights of $W^{(2)}$ of the computation graph of part (i), i.e., for each weight $w_{i,j}^{(2)}$ from a neuron i of the hidden layer to a neuron j of the output layer in the network of slides 18–19, we have shown in part (iv) that:

$$\frac{\partial E}{\partial w_{i,j}^{(2)}} = \frac{\partial E}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial w_{i,j}^{(2)}} = \frac{\partial E}{\partial s_j^{(2)}} o_i^{(1)}$$

From part (ii), we also know that:

$$\frac{\partial E}{\partial s_j^{(2)}} = \frac{\partial E}{\partial o_j^{(2)}} \frac{\partial o_j^{(2)}}{\partial s_j^{(2)}} = \frac{\partial E}{\partial o_j^{(2)}} \frac{\partial \sigma\left(s_j^{(2)}\right)}{\partial s_j^{(2)}} = \frac{\partial E}{\partial o_j^{(2)}} \sigma\left(s_j^{(2)}\right)\left(1 - \sigma\left(s_j^{(2)}\right)\right)$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}^{(2)}} = \frac{\partial E}{\partial s_j^{(2)}} o_i^{(1)} = \frac{\partial E}{\partial o_j^{(2)}} \sigma\left(s_j^{(2)}\right)\left(1 - \sigma\left(s_j^{(2)}\right)\right) o_i^{(1)} = \frac{\partial E}{\partial o_j^{(2)}} o_j^{(2)}\left(1 - o_j^{(2)}\right) o_i^{(1)}$$

From part (i) (and Exercise 7) we also know that:

$$\frac{\partial E}{\partial o_j^{(2)}} = o_j^{(2)} - t_j$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}^{(2)}} = \frac{\partial E}{\partial o_j^{(2)}} o_j^{(2)}\left(1 - o_j^{(2)}\right) o_i^{(1)} = \left(o_j^{(2)} - t_j\right) o_j^{(2)}\left(1 - o_j^{(2)}\right) o_i^{(1)}$$
$$= \left(o_j^{(2)} - t_j\right) o_j^{(2)}\left(1 - o_j^{(2)}\right) x_{i,j}$$

where $x_{i,j} = o_i^{(1)}$ is the signal from neuron i of the hidden layer to a neuron j of the output layer.

Therefore, the update rule for $w_{i,j}^{(2)}$ is:

$$w_{i,j}^{(2)} \leftarrow w_{i,j}^{(2)} - \eta \frac{\partial E}{\partial w_{i,j}^{(2)}} = w_{i,j}^{(2)} - \eta\left(o_j^{(2)} - t_j\right) o_j^{(2)}\left(1 - o_j^{(2)}\right) x_{i,j} =$$

$$= w_{i,j}^{(2)} + \eta\left(t_j - o_j^{(2)}\right) o_j^{(2)}\left(1 - o_j^{(2)}\right) x_{i,j}$$

which is the same as the update rule of Exercise 6 (and slide 21).

Let us now compute the update rule for the weights $W^{(1)}$ of the computation graph of part (i), i.e., for each weight $w_{i,j}^{(1)}$ from a neuron i of the input layer (that simply copies the input x_i) to a neuron j of the hidden layer in the network of slides 18–19. By reusing our conclusions of part (iv) for a matrix-vector multiplication node of the computation graph, we obtain:

$$\frac{\partial E}{\partial w_{i,j}^{(1)}} = \frac{\partial E}{\partial s_j^{(1)}} \frac{\partial s_j^{(1)}}{\partial w_{i,j}^{(1)}} = \frac{\partial E}{\partial s_j^{(1)}} x_i = \frac{\partial E}{\partial s_j^{(1)}} x_{i,j}$$

By reusing our conclusions of part (ii) for a sigmoid node of the computation graph, we also obtain:

$$\frac{\partial E}{\partial s_j^{(1)}} = \frac{\partial E}{\partial o_j^{(1)}} \frac{\partial o_j^{(1)}}{\partial s_j^{(1)}} = \frac{\partial E}{\partial o_j^{(1)}} \frac{\partial \sigma\left(s_j^{(1)}\right)}{\partial s_j^{(1)}} = \frac{\partial E}{\partial o_j^{(1)}} \sigma\left(s_j^{(1)}\right)\left(1 - \sigma\left(s_j^{(1)}\right)\right)$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}^{(1)}} = \frac{\partial E}{\partial s_j^{(1)}} x_{i,j} = \frac{\partial E}{\partial o_j^{(1)}} \sigma\left(s_j^{(1)}\right)\left(1 - \sigma\left(s_j^{(1)}\right)\right) x_{i,j} = \frac{\partial E}{\partial o_j^{(1)}} o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j}$$

By applying our conclusions of part (iii) to the rightmost matrix-vector multiplication node of the computation graph, we obtain:

$$\frac{\partial E}{\partial o_j^{(1)}} = \sum_k \frac{\partial E}{\partial s_k^{(2)}} \frac{\partial s_k^{(2)}}{\partial o_j^{(1)}} = \sum_k \frac{\partial E}{\partial s_k^{(2)}} w_{j,k}^{(2)}$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}^{(1)}} = \frac{\partial E}{\partial o_j^{(1)}} o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j} = \left(\sum_k \frac{\partial E}{\partial s_k^{(2)}} w_{j,k}^{(2)}\right) o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j}$$

By setting $\delta_k = -\frac{\partial E}{\partial s_k^{(2)}}$ *as in Exercise 6, we obtain:*

$$\frac{\partial E}{\partial w_{i,j}^{(1)}} = \left(\sum_k \frac{\partial E}{\partial s_k^{(2)}} w_{j,k}^{(2)}\right) o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j} = -\left(\sum_k \delta_k w_{j,k}^{(2)}\right) o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j}$$

Therefore, the update rule for $w_{i,j}^{(1)}$ *is:*

$$w_{i,j}^{(1)} \leftarrow w_{i,j}^{(1)} - \eta \frac{\partial E}{\partial w_{i,j}^{(1)}} = w_{i,j}^{(1)} + \eta \left(\sum_k \delta_k w_{j,k}^{(2)}\right) o_j^{(1)}\left(1 - o_j^{(1)}\right) x_{i,j}$$

which is the same as the update rule of Exercise 6 (and slide 21).

9. By working as in parts (i)–(iv) of Exercise 8 on the computation graph (not as in Exercise 6), derive the update rules for the weights $W^{(1)}$ and $W^{(2)}$ of the network of slide 32 (the same network as in Exercise 8, but without the rightmost sigmoid). You may use a sigmoid instead of *tanh* in the network of slide 32. You may reuse the conclusions of Exercises 7 and 8 for the backpropagation of gradients through matrix-vector multiplication, sigmoid, and squared error loss nodes.

10. Repeat Exercise 17 of Part 2 (Text Classification), now using a neural network (e.g., MLP, RNN, CNN), implemented using tools like Keras, TensorFlow, PyTorch, or DyNet.[5]

[5] See http://keras.io/, https://www.tensorflow.org/, http://web.stanford.edu/class/cs20si/, http://pytorch.org/. http://dynet.io/.