Assignment 3: Backpropagation Algorithm

MIE1516

In this document, we will guide you through the maths of the Backpropagation algorithm, which is actually nothing but the chain rule. You need to understand at least the final results of the derivation to be able to implement the code.

Let's first look at the notations used throughout this article.

Notations

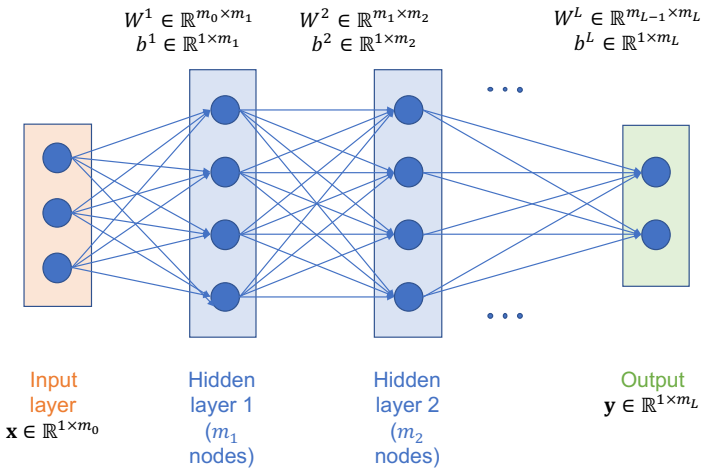

Figure 1: A Fully-Connected Neural Network Architecture

Look at Figure 1. We denote

- input sample: $\mathbf{x} \in \mathbb{R}^{1 \times m_0}$
- weights of lth layer: $W^l \in \mathbb{R}^{m_{l-1} \times m_l}$
- bias of lth layer: $b^l \in \mathbb{R}^{1 \times m_l}$
- non-linear activation function (element-wise): $\phi(\cdot)$
- the input to the lth layer: $u^l = h^{l-1}W^l + b^l \in \mathbb{R}^{1 \times m_l}$
- the output of lth layer: $h^l = \phi(u^l) \in \mathbb{R}^{1 \times m_l}$
- L is the total number of layers
- $\mathbf{t} \in \mathbb{R}^{1 \times m_L}$(single-input) or $\mathbf{t} \in \mathbb{R}^{N \times m_L}$ (multiple-input) is the target vector
- $\mathbf{y} \in \mathbb{R}^{1 \times m_L}$ (single-input) or $\mathbf{y} \in \mathbb{R}^{N \times m_L}$ (multiple-input) is the output of the network
- $E \in \mathbb{R}$ is the loss function:

- η is the learning rate in the gradient descent

  Note that $h^0 = \mathbf{x}$ and $h^L = \mathbf{y}$ by construction.

1 Single-Input Case

Let's assume that we only have a single input sample to feed forward and get the error from. The output $\mathbf{y}$ of the input can be expressed in a nested form:

$$\mathbf{y} = \phi\Big(((\dots)W^{l-1} + b^{l-1})W^l + b^l\Big)$$

Then, the loss E can be computed, and we get the derivative of E w.r.t. each and every parameter of the network by applying the Backpropagation algorithm. In other words, we want:

$$\frac{\partial E}{\partial W^l} \quad \text{and} \quad \frac{\partial E}{\partial b^l} \tag{1}$$

so that we can take the following updates:

$$W^l \leftarrow W^l - \eta\frac{\partial E}{\partial W^l} \quad \text{and} \quad b^l \leftarrow b^l - \eta\frac{\partial E}{\partial b^l} \tag{2}$$

Now, let's look at equation by equation to understand how we should go about deriving the derivative (1) w.r.t. each weight! (also see chapter 4 of Tom Mitchell's book: 'Machine Learning' (1996)). As done in this reference, it helps to look separately at the output nodes and the hidden nodes. Note that we start off with all the indices of weight matrices and vectors, but they will be converted to vectorized notations subsequently.

1.1 Output Layer

In the output nodes, we get the error signal directly from the loss function, E. So, the gradient of the error w.r.t. W^L and b^L are given as follow:

$$\frac{\partial E}{\partial W_{ij}^L} = \frac{\partial E}{\partial u_j^L}\cdot\frac{\partial u_j^L}{\partial W_{ij}^L} \qquad \left(\text{where} \quad \frac{\partial E}{\partial u_j^L} = \frac{\partial E}{\partial h_j^L}\cdot\frac{\partial h_j^L}{\partial u_j^L}\right) \tag{3}$$

$$\Rightarrow \frac{\partial E}{\partial W_{ij}^L} = \frac{\partial E}{\partial h_j^L}\cdot\frac{\partial h_j^L}{\partial u_j^L}\cdot\frac{\partial u_j^L}{\partial W_{ij}^L} \tag{4}$$

$$\text{Similarly,} \quad \frac{\partial E}{\partial b_j^L} = \frac{\partial E}{\partial h_j^L}\cdot\frac{\partial h_j^L}{\partial u_j^L}\cdot\frac{\partial u_j^L}{\partial b_j^L} \tag{5}$$

From equation (3) to equation (4), we are assuming that 1) the jth output node affects the loss only through the input u_j^L and that 2) h_j^L is determined by u_j^L alone. For now, we can

keep this assumption, but we will see that this is not the case when using the softmax output and the cross entropy loss.

Two things you need to see here are

1. W_{ij}^L affects the loss through u_j^L
2. $\frac{\partial u_j^L}{\partial W_{ij}^L}$ and $\frac{\partial u_j^L}{\partial b_j^L}$ reduce to:

$$\frac{\partial u_j^L}{\partial W_{ij}^L} = \frac{\partial(\sum_k h_k^{L-1} \cdot W_{kj}^L + b_j^L)}{\partial W_{ij}^L} = h_i^{L-1} \tag{6}$$

$$\frac{\partial u_j^L}{\partial b_j^L} = \frac{\partial(\sum_k h_k^{L-1} \cdot W_{kj}^L + b_j^L)}{\partial b_j^L} = 1 \tag{7}$$

Hence, we have $\frac{\partial E}{\partial W_{ij}^L} = \frac{\partial E}{\partial h_j^L}\frac{\partial h_j^L}{\partial u_j^L} h_i^{L-1} = \delta_j^L h_i^{L-1}$ and $\frac{\partial E}{\partial b_j^L} = \frac{\partial E}{\partial h_j^L}\frac{\partial h_j^L}{\partial u_j^L} = \delta_j^L$ where we have defined a crucial term δ as follows:

$$\delta_j^l = \frac{\partial E}{\partial u_j^l} \tag{8}$$

That is, the jth term of δ^l in the lth layer is the derivative of E w.r.t. the jth input (u_j^l). We will see that this term plays a key role as it is the one that's going to be passed down to the inner layers.

1.2 Hidden Layers

In the hidden units, we still want to compute the gradient of the loss w.r.t. each weight W_{ij}^l and bias b_j^l. However, unlike in the output nodes, now W_{ij}^l can affect E through multiple paths. It's helpful to look at the diagram below.

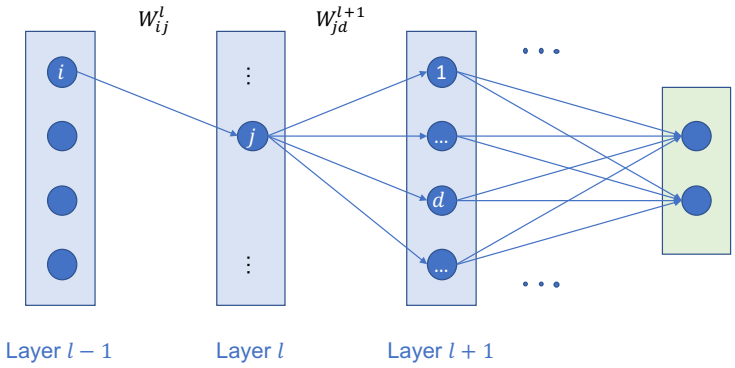

Figure 2: Connections in Hidden Layers

As can be seen, we need to consider the downstream nodes of jth node in the lth layer to correctly compute $\frac{\partial E}{\partial W_{ij}^l}$. However, it is still the case that E depends on W_{ij}^l only through u_j^l; hence, $\frac{\partial E}{\partial W_{ij}^l} = \frac{\partial E}{\partial u_j^l} \cdot \frac{\partial u_j^l}{\partial W_{ij}^l}$ holds.

Now, let's look at the δ:

$$\begin{aligned}\delta_j^l = \frac{\partial E}{\partial u_j^l} &= \sum_{d \in \mathcal{D}_j} \frac{\partial E}{\partial u_d^{l+1}} \frac{\partial u_d^{l+1}}{\partial u_j^l} = \sum_{d \in \mathcal{D}_j} \delta_d^{l+1} \cdot \frac{\partial u_d^{l+1}}{\partial u_j^l} \\ &= \sum_{d \in \mathcal{D}_j} \delta_d^{l+1} \frac{\partial u_d^{l+1}}{\partial h_j^l} \cdot \frac{\partial h_j^l}{\partial u_j^l} = \sum_{d \in \mathcal{D}_j} \delta_d^{l+1} W_{jd}^{l+1} \cdot \frac{\partial h_j^l}{\partial u_j^l} \end{aligned} \tag{9}$$

where the relation $\frac{\partial u_d^{l+1}}{\partial h_j^l} = \frac{\partial}{\partial h_j^l}\left(\sum_k h_k^l W_{kd}^{l+1} + b_d^{l+1}\right) = W_{jd}^{l+1}$ is used, and $\mathcal{D}_j$ is the set of downstream nodes of the jth node.

As a result,

$$\frac{\partial E}{\partial W_{ij}^l} = \delta_j^l \cdot \frac{\partial u_j^l}{\partial W_{ij}^l} = \left[\sum_d \delta_d^{l+1} W_{jd}^{l+1}\right] \frac{\partial h_j^l}{\partial u_j^l} \cdot \frac{\partial u_j^l}{\partial W_{ij}^l} = \left[\sum_d \delta_d^{l+1} W_{jd}^{l+1}\right] \frac{\partial h_j^l}{\partial u_j^l} h_i^{l-1} \tag{10}$$

Similarly, $$\frac{\partial E}{\partial b_j^l} = \frac{\partial E}{\partial u_j^l} \frac{\partial u_j^l}{\partial b_j^l} = \left[\sum_d \delta_d^{l+1} W_{jd}^{l+1}\right] \frac{\partial h_j^l}{\partial u_j^l} \frac{\partial u_j^l}{\partial b_j^l} = \left[\sum_d \delta_d^{l+1} W_{jd}^{l+1}\right] \frac{\partial h_j^l}{\partial u_j^l} \tag{11}$$

where $\frac{\partial u_j^l}{\partial W_{ij}^l} = h_i^{l-1}$ and $\frac{\partial u_j}{\partial b_j^l} = 1$.

To sum up this part, we have derived the expression for the gradient of the loss w.r.t. every weight and bias beginning from the output layer all the way down to the first layer. Clearly, δ^l receives the gradient information from the layer $l+1$ through δ^{l+1}, W_{jd}^{l+1}, and the activation (see equation (9)).

1.3 Vectorized Notation

It is crucial to be able to write all the expressions derived so far in a vectorized form if we are to implement an efficient neural network module using NumPy. So, let's convert the summations into matrix multiplications and element-wise multiplications.

1. Output Layer

$$\begin{aligned}\frac{\partial E}{\partial W^L} &= \begin{pmatrix} \frac{\partial E}{\partial W_{11}^L} & \frac{\partial E}{\partial W_{12}^L} & \cdots \\ \frac{\partial E}{\partial W_{21}^L} & \frac{\partial E}{\partial W_{22}^L} & \cdots \\ \vdots & \vdots & \cdots \end{pmatrix} = \begin{pmatrix} h_1^{L-1} \frac{\partial E}{\partial h_1^L} \frac{\partial h_1^L}{\partial u_1^L} & h_1^{L-1} \frac{\partial E}{\partial h_2^L} \frac{\partial h_2^L}{\partial u_2^L} & \cdots \\ h_2^{L-1} \frac{\partial E}{\partial h_1^L} \frac{\partial h_1^L}{\partial u_1^L} & h_2^{L-1} \frac{\partial E}{\partial h_2^L} \frac{\partial h_2^L}{\partial u_2^L} & \cdots \\ \vdots & \vdots & \cdots \end{pmatrix} \in \mathbb{R}^{m_{L-1} \times m_L} \\ &= (h^{L-1})^\top \left[\frac{\partial E}{\partial h^L} \odot \phi'(u^L)\right] = (h^{L-1})^\top \delta^L \end{aligned} \tag{12}$$

$$\frac{\partial E}{\partial b^L} = \frac{\partial E}{\partial h^L} \odot \phi'(u^L) = \delta^L \tag{13}$$

where $\delta^L = \frac{\partial E}{\partial u^L} = \frac{\partial E}{\partial h^L} \odot \phi'(u^L)$ and $\phi'(u^L) = \frac{\partial \phi(u^L)}{\partial u^L}$, with $\odot$ denoting element-wise multiplication.

2. Hidden Layers

$$\frac{\partial E}{\partial W^l} = (h^{l-1})^\top \left[\delta^{l+1} (W^{l+1})^\top \odot \phi'(u^l) \right] \tag{14}$$

$$\frac{\partial E}{\partial b^l} = \delta^{l+1} (W^{l+1})^\top \odot \phi'(u^l) \tag{15}$$

3. Delta:

$$\delta^l = [\delta^{l+1} W^{l+1}] \odot \phi'(u^l) \tag{16}$$

So far, we've only worked with a single input sample, which may be used in the pure SGD implementation. However, what we really want is the vectorized notations for mini-batch SGD.

2 Multiple-Input Case

Now, we need to introduce uppercase notations for multiple inputs.

- The number of input samples: N
- Input data matrix: $X \in \mathbb{R}^{N \times m_0}$
- The input to the lth layer: $U^l = H^{l-1} W^l + \mathbf{1} b^l \in \mathbb{R}^{N \times m_l}$ where $\mathbf{1} \in \mathbb{R}^{N \times 1}$ is a column vector of ones and $b^l \in \mathbb{R}^{1 \times m_l}$ is now a row vector.
- The output of the lth layer: $H^l = \phi(U^l) \in \mathbb{R}^{N \times m_l}$

We normally have the loss function that can be factorized over each sample, that is, $E = \sum_{n=1}^{N} E^n$ where E^n is the loss generated from the nth sample. So, we just need to modify previously developed notations while paying careful attention to the dimension of each matrix or vector. Let's begin with the output layer.

2.1 Output Layer

$$\frac{\partial E}{\partial W^L} = \sum_n \frac{\partial E^n}{\partial W^L} = \begin{bmatrix} H_{11}^{L-1}\frac{\partial E^1}{\partial H_{11}^L}\frac{\partial H_{11}^L}{\partial U_{11}^L} & H_{11}^{L-1}\frac{\partial E^1}{\partial H_{12}^L}\frac{\partial H_{12}^L}{\partial U_{12}^L} & \cdots \\ H_{12}^{L-1}\frac{\partial E^1}{\partial H_{11}^L}\frac{\partial H_{11}^L}{\partial U_{11}^L} & H_{12}^{L-1}\frac{\partial E^1}{\partial H_{12}^L}\frac{\partial H_{12}^L}{\partial U_{12}^L} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} + \begin{bmatrix} H_{21}^{L-1}\frac{\partial E^2}{\partial H_{21}^L}\frac{\partial H_{21}^L}{\partial U_{21}^L} & H_{21}^{L-1}\frac{\partial E^2}{\partial H_{22}^L}\frac{\partial H_{22}^L}{\partial U_{22}^L} & \cdots \\ H_{22}^{L-1}\frac{\partial E^2}{\partial H_{21}^L}\frac{\partial H_{21}^L}{\partial U_{21}^L} & H_{22}^{L-1}\frac{\partial E^2}{\partial H_{22}^L}\frac{\partial H_{22}^L}{\partial U_{22}^L} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} + \cdots$$

$$= \begin{bmatrix} \sum_n H_{n1}^{L-1}\frac{\partial E^n}{\partial H_{n1}^L}\frac{\partial H_{n1}^L}{\partial U_{n1}^L} & \sum_n H_{n1}^{L-1}\frac{\partial E^n}{\partial H_{n2}^L}\frac{\partial H_{n2}^L}{\partial U_{n2}^L} & \cdots \\ \sum_n H_{n2}^{L-1}\frac{\partial E^n}{\partial H_{n1}^L}\frac{\partial H_{n1}^L}{\partial U_{n1}^L} & \sum_n H_{n2}^{L-1}\frac{\partial E^n}{\partial H_{n2}^L}\frac{\partial H_{n2}^L}{\partial U_{n2}^L} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

When we look at the ij component of the above gradient:
$\left(\frac{\partial E}{\partial W^L}\right)_{ij} = \sum_n H_{ni}^{L-1}\frac{\partial E^n}{\partial H_{nj}^L}\frac{\partial H_{nj}^L}{\partial U_{nj}^L} = \sum_n H_{ni}^{L-1}\delta_{nj}^L$

where we can see that $\delta_{nj}^L = \frac{\partial E^n}{\partial U_{nj}^L} = \frac{\partial E^n}{\partial H_{nj}^L}\frac{\partial H_{nj}^L}{\partial U_{nj}^L} = \frac{\partial E^n}{\partial H_{nj}^L}\phi'(U_{nj}^L)$, or in vectorized form, $\delta_{n\cdot}^L = \frac{\partial E^n}{\partial H_{n\cdot}^L} \odot \phi'(U_{n\cdot}^L)$, which is a $(1 \times m_L)$ vector for a single sample n. Hence, in the matrix form, we get below:

$$\frac{\partial E}{\partial W^L} = (H^{L-1})^\top \delta^L \tag{17}$$

where $\delta^L \in \mathbb{R}^{n \times m_L}$.

Similarly, the jth term of the gradient w.r.t. the bias is:

$$\begin{aligned} \left(\frac{\partial E}{\partial b^L}\right)_j &= \sum_n \frac{\partial E^n}{\partial U_{nj}^L}\frac{\partial U_{nj}^L}{\partial b_j^L} = \sum_n \delta_{nj}^L \\ &= (\mathbf{1}^\top \delta^L)_j \\ \text{or,} \quad \frac{\partial E}{\partial b^L} &= \mathbf{1}^\top \delta^L \end{aligned} \tag{18}$$

2.2 Hidden Layers

Finally, for the hidden layer, we have the following:

$$\begin{aligned}
\left(\frac{\partial E}{\partial W^l}\right)_{ij} &= \left(\sum_n \frac{\partial E^n}{\partial W^l}\right)_{ij} = \sum_n \frac{\partial E^n}{\partial W^l_{ij}} = \sum_n \Big[\sum_d \delta^{l+1}_{nd} W^{l+1}_{jd}\Big] \frac{\partial H^l_{nj}}{\partial U^l_{nj}} H^{l-1}_{ni} \\
&= \sum_n H^{l-1}_{ni} \delta^{l+1}_{n\cdot} (W^{l+1}_{j\cdot})^\top \frac{\partial H^l_{nj}}{\partial U^l_{nj}} = \sum_n H^{l-1}_{ni} \big[\delta^{l+1}(W^{l+1})^\top\big]_{nj} \phi'(U^l_{nj}) \\
&= \Big[(H^{l-1})^\top \Big\{\phi'(U^l) \odot \big[\delta^{l+1}(W^{l+1})^\top\big]\Big\}\Big]_{ij} \\
\therefore \frac{\partial E}{\partial W^l} &= (H^{l-1})^\top \Big\{\ \phi'(U^l) \odot \big[\delta^{l+1}(W^{l+1})^\top\big]\Big\} \qquad (19)
\end{aligned}$$

$$\text{and for the bias, } \frac{\partial E}{\partial b^l} = \mathbf{1}^\top \delta^l \qquad (20)$$

As in the single sample case, we need to express δ^l in relation to δ^{l+1} so that the gradient from the output layer can propagate back to hidden layers.

$$\text{In the output layer, } \delta^L_{n\cdot} = \frac{\partial E^n}{\partial H^L_{n\cdot}} \odot \phi'(U^L_{n\cdot}) \qquad (21)$$

$$\begin{aligned}
\text{in the hidden layers, } \delta^l_{nj} &= \frac{\partial E^n}{\partial U^l_{nj}} = \sum_{d\in\mathcal{D}_j} \frac{\partial E^n}{\partial U^{l+1}_{nd}} \frac{\partial U^{l+1}_{nd}}{\partial U^l_{nj}} = \sum_{d\in\mathcal{D}_j} \frac{\partial E^n}{\partial U^{l+1}_{nd}} W^{l+1}_{jd} \phi'(U^l_{nj}) \\
&= \big[\big\{\delta^{l+1}(W^{l+1})^\top\big\} \odot \phi'(U^l)\big]_{nj} \\
\text{or, } \delta^l &= \big\{\delta^{l+1}(W^{l+1})^\top\big\} \odot \phi'(U^l) \qquad (22)
\end{aligned}$$

3 Loss, Activation Functions and Their Derivatives

Now that we have the vectorized equations of gradients, we are almost there to actually implement a working neural network module. The remaining part is to derive the derivative of a loss function w.r.t. the output as well as the derivative of the activation function ($\phi(U^l)$) w.r.t. its input (U^l).

3.1 Loss Functions

In this part, let's restrict our attention to the mean-squared error (MSE) loss and the cross entropy loss.

1. MSE loss

This is one of the most commonly used losses (e.g. in the least-squares method), which is defined as follows:

$$E = \frac{1}{2N}\sum_{n=1}^{N}\|\mathbf{t}_n - \mathbf{y}_n\|^2 = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j=1}^{m_L}(t_{nj} - y_{nj})^2$$

where $\mathbf{t}_n \in \mathbb{R}^{1\times m_L}$ is the nth target value and $\mathbf{y}_n$ is the corresponding output from the neural net.

Then, computing the derivative of E w.r.t. y is easy:

$$\begin{aligned}\frac{\partial E^n}{\partial y_{nj}} &= \frac{1}{N}(y_{nj} - t_{nj}) \\ \delta^L_{nj} = \frac{\partial E}{\partial y_{nj}}\frac{\partial y_{nj}}{\partial U^L_{nj}} &= \frac{1}{N}(y_{nj} - t_{nj})\cdot\phi'(U^L_{nj}) \\ \therefore \quad \delta^L &= \frac{1}{N}(\mathbf{y} - \mathbf{t})\odot\phi'(U^L) \end{aligned} \tag{23}$$

Normally in a regression problem, we want the outputs to have a unbounded range, so we can simply use the linear output without an activation (i.e., $\mathbf{y} = U^L$). In a classification problem, however, we want the outputs to be within 0 and 1 to represent probabilities of being certain classes, so the softmax activation function is applied to the linear output (i.e., $\mathbf{y} = \phi(U^L)$).

2. Cross-entropy loss

   While the MSE loss is normally used in a regression problem, the cross-entropy loss is preferred when working with classification. Since the cross-entropy loss takes probabilities as its inputs, we need to squash output values from the output layer such that $\sum_j H^L_{nj} = 1$ as well as $H^L_{nj} \in [0, 1]$ for all n and j. Hence, the activation function in the output layer is set to the softmax function. The loss is defined as follows:

$$\begin{aligned} E &= \sum_{n=1}^{n} E^n \\ \text{where} \quad E^n &= -\sum_{j=1}^{m_L} t_{nj}\log(y_{nj}) \end{aligned}$$

   Using the cross-entropy loss along with the softmax output changes one of the core assumptions that we made when deriving the gradients (see equation 3), so we need to take care of this case separately.

3.2 Activation Functions

An activation function plays an indispensable role in a neural network since it is the only part where nonlinearity is introduced. Below listed are some commonly used activation functions along with their derivatives w.r.t. inputs.

1. ReLU (Rectified Linear Unit) and its derivative

   $\phi(U) \coloneqq \max(0, U)$

   $\phi'(U) = \begin{cases} 1, & \text{if } U > 0, \\ 0, & \text{otherwise} \end{cases}$

2. Sigmoid function and its derivative

   $\phi(U) \coloneqq \frac{1}{1+\exp(-U)}$

   $\phi'(U) = \phi(U)\big(1 - \phi(U)\big)$

3. Tangent hyperbolic function (tanh) and its derivative

   $\phi(U) \coloneqq \tanh(U) = \frac{\exp(U)-\exp(-U)}{\exp(U)+\exp(-U)}$

   $\phi'(U) = 1 - \tanh^2(U)$

4. Softmax function and the Cross-entropy loss

   Let's work out the derivative of the cross-entropy loss with the softmax output. Firstly, the softmax function is defined as follows:

   $$y_{nj} = \phi_{nj} = \frac{\exp(U^L_{nj})}{\sum_k \exp(U^L_{nk})}$$

   where y_{nj} is the jth component of the nth output $\mathbf{y}_n$.

   What changes when using the softmax output is δ because now we can't write $\frac{\partial E}{\partial U^L_{nj}} = \frac{\partial E}{\partial y_{nj}} \frac{\partial y_{nj}}{\partial U^L_{nj}}$. Instead, the input of the jth output node U^L_{nj} affects E through all the other nodes, which should be taken into account in the chain rule.

   $$\begin{aligned} \frac{\partial y_{nc}}{\partial U^L_{nj}} &= \begin{cases} y_{nj}(1 - y_{nj}) & \text{if } \quad c = j, \\ -y_{nc} \cdot y_{nj} & \text{otherwise} \end{cases} \\ &= y_{nc}(1_{c=j} - y_{nj}) \end{aligned}$$

   where $1_{c=j}$ is the indicator function (or direc-delta function) that gives 1 when $c = j$, and 0 otherwise.

The derivative of the cross-entropy loss w.r.t. the njth output is $\frac{\partial E}{\partial y_{nj}} = -\frac{t_{nj}}{y_{nj}}$; hence, the delta becomes

$$\begin{aligned}
\delta^L_{nj} = \frac{\partial E}{\partial U^L_{nj}} &= \sum_c \frac{\partial E}{\partial y_{nc}} \frac{\partial y_{nc}}{\partial U^L_{nj}} \\
&= \sum_c \frac{\partial E}{\partial y_{nc}} y_{nc}(1_{c=j} - y_{nj}) \\
&= \frac{\partial E^n}{\partial y_{nj}} y_{nj} - \sum_c \frac{\partial E^n}{\partial y_{nc}} y_{nc} y_{nj} \\
&= -\frac{t_{nj}}{y_{nj}} y_{nj} - \sum_c \left(-\frac{t_{nc}}{y_{nc}} \right) y_{nc} y_{nj} \\
&= y_{nj} \sum_c t_{nc} - t_{nj} = y_{nj} - t_{nj} \\
\text{or,} \quad \delta^L &= \mathbf{y} - \mathbf{t} \qquad (24)
\end{aligned}$$

where I used the fact that $t_{n\cdot}$ is one-hot encoded.

That's all you have to know to understand the neural network implementation! Now, go ahead and play with the module :)