Neural Networks
Example gradient calculation

Consider a two-layer network with K hidden units and total output

$$\sigma(\boldsymbol{\xi}) = h\left(\sum_{j=1}^{K} v_j\, g(\boldsymbol{w}^{(j)} \cdot \boldsymbol{\xi})\right) \tag{1}$$

Note that this is the same as in the lecture notes (page 3, Gradient Based Learning), but I am using the clearer notation $\boldsymbol{w}^{(j)}$ instead of $\boldsymbol{w}_j$ for the j-th input-to-hidden weight vector.
We are interested in the usual quadratic cost function

$$E = \sum_{\mu=1}^{P} e^{\mu} \quad \text{with single example terms} \quad e^{\mu} = \frac{1}{2}\left(\sigma(\boldsymbol{\xi}^{\mu}) - \tau(\boldsymbol{\xi}^{\mu})\right)^2 \tag{2}$$

Note that only σ depends on the weights, the target values τ are fixed and given in the data set. In the following we consider the gradient of only one example term e^{μ}. For convenience, we omit the index μ for a while and write σ in short for $\sigma(\boldsymbol{\xi})$ etc.
First, the derivative with respect to one of the hidden-to-output weights:

$$\frac{\partial e}{\partial v_k} = (\sigma - \tau)\ \frac{\partial \sigma}{\partial v_k} = \underbrace{(\sigma - \tau)\ \ h'\left(\sum_{j=1}^{K} v_j\, g(\boldsymbol{w}^{(j)} \cdot \boldsymbol{\xi})\right)}_{\text{shorthand: } \delta}\ g(\boldsymbol{w}^{(k)} \cdot \boldsymbol{\xi}) \tag{3}$$

Hint: use an index (here:k) for the variable with respect you differentiate that is different from all indices used in the definition of e. Otherwise, confusion is unavoidable.

Of course h' can be worked out in a practical example. For instance: if $h(x) = \tanh(\gamma x)$, then $h'(x) = \gamma\left(1 - \tanh^2(\gamma x)\right)$.
Note that g does not depend on any of the v_j, so we don't have to take more (inner) derivatives, here.
Now we take the derivative with respect to a single input-to-hidden weight $w_n^{(m)}$,

i.e. the n-th component of the m-th weight vector. Note that the same δ as defined above appears here, too:

$$\frac{\partial e}{w_n^{(m)}} = (\sigma - \tau)\ \frac{\partial \sigma}{\partial w_n^{(m)}} = \delta\ v_m\, g'\left(\boldsymbol{w}^{(m)} \cdot \boldsymbol{\xi}\right)\ \xi_n \tag{4}$$

The new term $v_m\, g'(\ldots)$ corresponds to the derivative of the only term in the sum $\sum_{j=1}^{K} \ldots$ that actually contains the weight vector $\boldsymbol{w}^{(m)}$. Finally the factor ξ_n appears because

$$\boldsymbol{w}^{(m)} \cdot \boldsymbol{\xi} = \sum_{i=1}^{N} \boldsymbol{w}_i^{(m)} \xi_i \quad \text{and thus} \quad \frac{\partial}{w_n^{(m)}}(\boldsymbol{w}^{(m)} \cdot \boldsymbol{\xi}) = \xi_n.$$

It is always helpful to make a plausibility check: derivatives with respect to single weights are just numbers, so the right hand sides of Eqs. (3) and (4) should be numbers too (not vectors, for instance). This is fine, it seems.
Now, of course, we could build gradient vectors from the component-wise results. Formally, we could combine all weights in one long vector, but we would have to specify the updates separately for the v_j and $\boldsymbol{w}^{(k)}$, anyway. Here, it only makes sense to write for $m = 1, \dots K$:

$$\nabla_{w^{(m)}} e = \delta \, v_m \, g'\left(\boldsymbol{w}^{(m)} \cdot \boldsymbol{\xi}\right) \, \boldsymbol{\xi} \tag{5}$$

where the notation $\nabla_{w^{(m)}}$ stands for *the gradient with respect to the m-th weight vector*. Both, left and right hand sides of Eq. (5) are N-dimensional vectors!
For the gradient of the full cost function we have to perform sums over examples again (note that δ is also defined for a particular example):

$$\frac{\partial}{\partial v_k} E = \sum_{\mu=1}^{P} \underbrace{\left(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)\right) \; h'\left(\sum_{j=1}^{K} v_j \, g(\boldsymbol{w}^{(j)} \cdot \boldsymbol{\xi}^\mu)\right)}_{\text{shorthand: } \delta^\mu} \; g(\boldsymbol{w}^{(k)} \cdot \boldsymbol{\xi}^\mu) \tag{6}$$

$$\nabla_{w^{(m)}} E = \sum_{\mu=1}^{P} \delta^\mu \, v_m \, g'\left(\boldsymbol{w}^{(m)} \cdot \boldsymbol{\xi}^\mu\right) \, \boldsymbol{\xi}^\mu \tag{7}$$

Do not forget that in gradient <u>descent</u>, of course, steps in the direction of the <u>negative</u> gradient have to be performed and a step size has to be specified.

Remark:
In a network with more layers, the chain rule has to be applied several times, and in every layer terms similar to δ from previous layers are re-used together with weights as coefficients. In a particular efficient implementation, one can go *down* the network to calculate the output σ and *up* again to calculate the gradient. Both ways, the same weights appear as coefficients in the calculation.
This is the idea behind the famous *backpropagation of error*. Originally, the term was coined for the efficient computation of the gradient. Nowadays, it is mostly used for the entire gradient based training procedure.