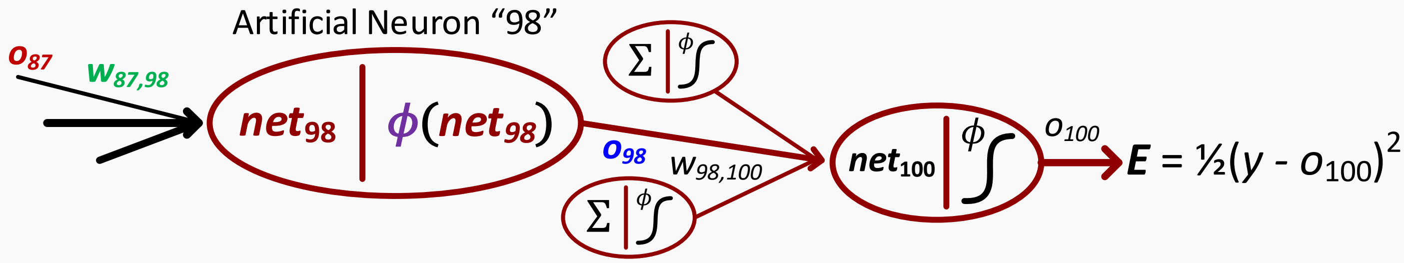
backpropagation through neural network hidden layer

The following illustrates calculations passing back through an **Artificial Neural Network** layer:



Below represents the derivative of error $\boldsymbol{E}$ using the **chain rule** from calculus:

$$\frac{dE}{d\boldsymbol{w}_{87,98}} = \frac{dE}{d\boldsymbol{o}_{98}} \frac{d\boldsymbol{o}_{98}}{d\boldsymbol{net}_{98}} \frac{d\boldsymbol{net}_{98}}{d\boldsymbol{w}_{87,98}} \qquad \boldsymbol{\phi}(z) = \frac{1}{1+e^{z}}$$

$$\boldsymbol{\phi}'^{(z)} = \frac{d\boldsymbol{\phi}(z)}{dz} = \boldsymbol{\phi}(z)\big(1-\boldsymbol{\phi}(z)\big)$$

Computing the **last** derivative term:

$$\frac{d\boldsymbol{net}_{98}}{d\boldsymbol{w}_{87,98}} = \frac{d\big(\boldsymbol{w}_{87,98}\boldsymbol{o}_{87} + w_{86,98}o_{86} + w_{85,98}o_{85} + \cdots\big)}{d\boldsymbol{w}_{87,98}} = \boldsymbol{o}_{87}$$

Computing the **middle** derivative term:

$$\frac{d\boldsymbol{o}_{98}}{d\boldsymbol{net}_{98}} = \frac{d\boldsymbol{\phi}(\boldsymbol{net}_{98})}{d\boldsymbol{net}_{98}} = \boldsymbol{\phi}'(\boldsymbol{net}_{98})\big(1-\boldsymbol{\phi}(\boldsymbol{net}_{98})\big) = \boldsymbol{o}_{98}(1-\boldsymbol{o}_{98})$$

Computing the **first** derivative term:

$$\frac{dE}{d\boldsymbol{o}_{98}} = \frac{dE}{d\boldsymbol{net}_{100}} \frac{d\boldsymbol{net}_{100}}{d\boldsymbol{o}_{98}}$$

$$\textbf{since } \frac{dE}{d\boldsymbol{net}_{100}} = \boldsymbol{\delta}_{100} \textbf{ and } \frac{d\boldsymbol{net}_{100}}{d\boldsymbol{o}_{98}} = \frac{d\big(w_{99,100}o_{99} + w_{98,100}\boldsymbol{o}_{98} + \big)}{d\boldsymbol{o}_{98}} = \boldsymbol{w}_{98,100}$$

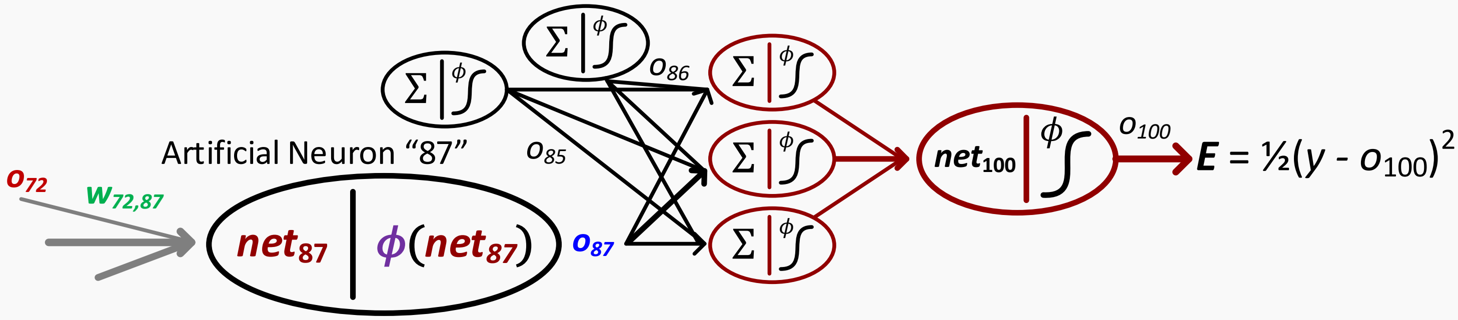
$$\textbf{then } \frac{dE}{d\boldsymbol{o}_{98}} = \boldsymbol{\delta}_{100}\boldsymbol{w}_{98,100}$$

The **result** of the calculation above is as follows:

$$\frac{dE}{d\boldsymbol{w}_{87,98}} = \frac{dE}{d\boldsymbol{o}_{98}} \frac{d\boldsymbol{o}_{98}}{d\boldsymbol{net}_{98}} \frac{d\boldsymbol{net}_{98}}{d\boldsymbol{w}_{87,98}} = \boldsymbol{\delta}_{100}\boldsymbol{w}_{98,100}\boldsymbol{o}_{98}(1-\boldsymbol{o}_{98})\boldsymbol{o}_{87}$$

Backpropagation through another Hidden Layer in a Neural Network

The following illustrates calculations passing back through another **Artificial Neural Network** layer:



Below represents the derivative of error $\boldsymbol{E}$ using the **chain rule** from calculus:

$$\frac{dE}{d\boldsymbol{w}_{72,87}} = \frac{dE}{d\boldsymbol{o}_{87}} \frac{d\boldsymbol{o}_{87}}{d\boldsymbol{net}_{87}} \frac{d\boldsymbol{net}_{87}}{d\boldsymbol{w}_{72,87}} \qquad \boldsymbol{\phi}(z) = \frac{1}{1+e^z}$$

$$\boldsymbol{\phi}'^{(z)} = \frac{d\boldsymbol{\phi}(z)}{dz} = \boldsymbol{\phi}(z)\left(1 - \boldsymbol{\phi}(z)\right)$$

Computing the **last** derivative term:

$$\frac{d\boldsymbol{net}_{87}}{d\boldsymbol{w}_{72,87}} = \frac{d\left(\boldsymbol{w}_{72,87}\boldsymbol{o}_{72} + w_{71,87}o_{71} + w_{70,87}o_{70} + \cdots\right)}{d\boldsymbol{w}_{72,87}} = \boldsymbol{o}_{72}$$

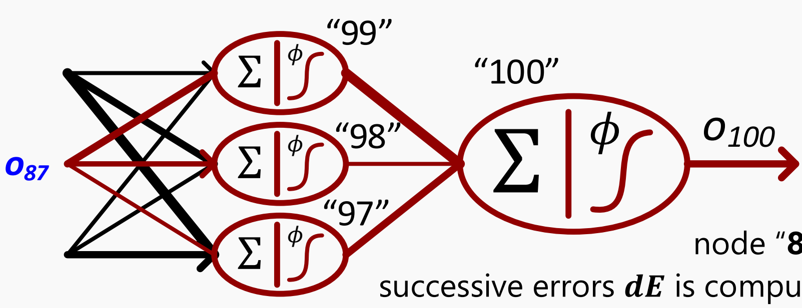
Computing the **middle** derivative term:

$$\frac{d\boldsymbol{o}_{87}}{d\boldsymbol{net}_{87}} = \frac{d\boldsymbol{\phi}(\boldsymbol{net}_{87})}{d\boldsymbol{net}_{87}} = \boldsymbol{\phi}'(\boldsymbol{net}_{87})\left(1 - \boldsymbol{\phi}(\boldsymbol{net}_{87})\right) = \boldsymbol{o}_{87}(1 - \boldsymbol{o}_{87})$$

Computing the **first** derivative term:

$$\frac{dE}{d\boldsymbol{o}_{87}} = \frac{dE}{d\boldsymbol{net}_{99}} \frac{d\boldsymbol{net}_{99}}{d\boldsymbol{o}_{87}} + \frac{dE}{d\boldsymbol{net}_{98}} \frac{d\boldsymbol{net}_{98}}{d\boldsymbol{o}_{87}} + \frac{dE}{d\boldsymbol{net}_{97}} \frac{d\boldsymbol{net}_{97}}{d\boldsymbol{o}_{87}} + \cdots$$

As the layers become deeper in the **Artificial Neural Network**, the computation of the **first derivative term** in calculating the weight for a given node's output becomes increasingly complex. This is due to the output of nodes within the hidden layer being dependent of the derivative of the error $\boldsymbol{dE}$ of all successive nodes to the final output $\boldsymbol{o}_{100}$ as computed and illustrated below:

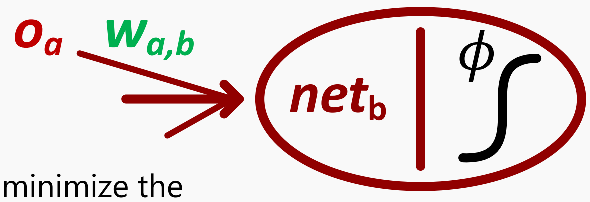


$$\delta_{99}w_{87,99} + \delta_{98}w_{87,98} + \delta_{97}w_{87,97} + \cdots$$
$$= \sum_{\ell \in L} \boldsymbol{\delta}_{87}\boldsymbol{w}_{87,\ell}$$

Therefore, the derivative of the output of node "**87**" → $\boldsymbol{o}_{87}$ with respect to the derivative of all successive errors $\boldsymbol{dE}$ is computed as the sum of all delta terms from the prior calculations in the **Backpropagation Algorithm** → $\sum_{\ell \in L} \boldsymbol{\delta}_{87}$ times the connectivity (**weight**) of the node "**87**" downstream in the current layer→ $\boldsymbol{w}_{87,\ell}$.

Writing the above computation in a more general way to understand **Backpropagation Intuition**:

$$\frac{dE}{dw_{a,b}} = \frac{dE}{do_b}\,\frac{do_b}{dnet_b}\,\frac{dnet_b}{dw_{a,b}}$$



Assuming Neuron "***a***" connected to Neuron "***b***". The **optimization objective** is to adjust the **weight** $w_{a,b}$ to minimize the derivative of the error with respect to the **weight** $\frac{dE}{dw_{a,b}}$; using the **chain rule**:

$$\frac{dE}{dw_{a,b}} = \frac{dE}{do_b}\,\frac{do_b}{dnet_b}\,\frac{dnet_b}{dw_{a,b}} = \frac{dE}{do_b}\,\frac{do_b}{dnet_b}\;o_a$$

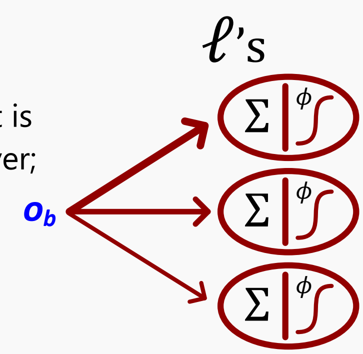
The **last** derivative term is always computed as the output of the prior node o_a.

$$\frac{dE}{dw_{a,b}} = \frac{dE}{do_b}\,\frac{do_b}{dnet_b}\,\frac{dnet_b}{dw_{a,b}} = \frac{dE}{do_b}\;o_b(1-o_b)o_a$$

The **middle** derivative term is always computed as the derivative of phi $\boldsymbol{\phi} \rightarrow\ o_b(1-o_b)$.

The **first** derivative term is the sum of the deltas δ from all of the downstream nodes and the downstream weights $\left(\sum_{\ell\in L}\delta_\ell\, w_{b,\ell}\right)$

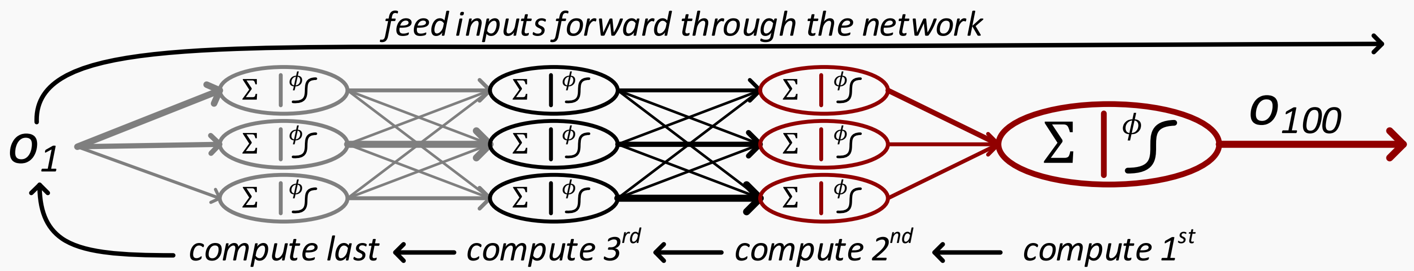
$$\frac{dE}{dw_{a,b}} = \frac{dE}{do_b}\,\frac{do_b}{dnet_b}\,\frac{dnet_b}{dw_{a,b}} = \left(\sum_{\ell\in L}\delta_\ell\, w_{b,\ell}\right)o_b(1-o_b)o_a$$



It is important that the L's represent the indices of **downstream neurons**. It is necessary to compute all of the δ_ℓ's subsequent to the current node in a layer; working backward along the neural network in order to compute. The computation as a whole represents the **gradient**:

$$\left(\sum_{\ell\in L}\delta_\ell\, w_{b,\ell}\right)o_b(1-o_b)o_a \rightarrow \textbf{Gradient Computation}$$

In other words, the implication of the above intuition is that the **gradients** are computed with respect to the **final output layer**, then the **gradients** are computed with respect to the **second last hidden layer**, then the **gradients** are computed with respect to the **third last hidden layer**, and so on… using the deltas $\boldsymbol{\delta}$**'s** each time that were computed downstream of the current layer.



Given an understanding to compute the derivative $\boldsymbol{dE}$ with respect to the weight $\boldsymbol{w_{a,b}}$ for all weights $\boldsymbol{w_{a,b}}$'s, the following explains the mathematics behind **gradient descent**:

$$\boldsymbol{w_{a,b}} \longleftarrow \boldsymbol{w_{a,b}} - \boldsymbol{\alpha} \frac{dE}{d\boldsymbol{w_{a,b}}}$$

Each weight $\boldsymbol{w_{a,b}}$ is taken, with a **step-sized** $\boldsymbol{\alpha}$ **weight** $\boldsymbol{w_{a,b}}$ taken down the **gradient**.

In standard **Backpropagation**, α is between **0** and **1** and referred to as the **learning rate**.

If the **learning rate** $\boldsymbol{\alpha}$ is too **low**, the neural network will learn very slowly. Conversely, if the **learning rate** $\boldsymbol{\alpha}$ is too **high**, the weights can cause an objective to **diverge** from the **optimal minimum**.

The general logic is that the weight changes depending on how **steep** the **gradient** is at that point.

Once the **errors** are **propagated** back through the neural network and the **weights** have been updated, the **error** must be updated; achieved by feeding the **input forward** through the network.

Neural Network Summarized

Backpropagation: Repeat the process going **backwards** (**gradient** calculation), adjusting the **weights**, and feeding the **input forward** (**error** calculation) by many interactions for **learning**

Advantages of Neural Networks

- Highly expressive nonlinear models
- Advanced in computer vision and speech that is unmatched through competing methods.
- Capable of capturing latent structure with the hidden network layers

Disadvantages of Neural Networks

- Prone to becoming stagnant in local optima and produce poor solutions
- Another black-box algorithm that requires extensive parameter tuning (network structure).