COMS 4030A
Adaptive Computation and Machine Learning

2.8. **Training Neural Networks.**

We describe here a method for training the weights of a neural network so that the network produces the correct output for its inputs. By 'weights' we mean both the edge weights and the bias values.

We require the notion of a **loss function**, which computes the loss, or error, between the actual output of the network and the target. There are a number of different loss functions.

Given a network with k output nodes, suppose an input $\boldsymbol{x}$ is fed forward through the network and the output $\boldsymbol{y} = (y_1, \ldots, y_k)$ is obtained. If $\boldsymbol{t} = (t_1, \ldots, t_k)$ is the target corresponding to the input $\boldsymbol{x}$, then a loss function L is a function of $\boldsymbol{y}$ and $\boldsymbol{t}$, written as $L(\boldsymbol{y}, \boldsymbol{t})$.
Note that the output $\boldsymbol{y}$ is a function of the input $\boldsymbol{x}$ and the weights of the network.

A commonly used loss function is the **sum-of-squares** loss function which is defined by:

$$L(\boldsymbol{y}, \boldsymbol{t}) = \tfrac{1}{2}\sum_{j=1}^{k}(y_j - t_j)^2$$

The above expression gives the loss for a single input $\boldsymbol{x}$. Loss can also be calculated over a set of input values by taking the sum of all the single losses.
Note that if the output $\boldsymbol{y}$ of the network is the same as the target $\boldsymbol{t}$, then $L(\boldsymbol{y}, \boldsymbol{t}) = 0$.

Example: Suppose that for some network with three output nodes, an input $\boldsymbol{x}$ has a target of $\boldsymbol{t} = (1, 0, 0)$ and the output from the network is $\boldsymbol{y} = (0.7, 0.3, 0.4)$. Then the sum-of-squares loss for the input $\boldsymbol{x}$ is:

$$L(\boldsymbol{y}, \boldsymbol{t}) = \tfrac{1}{2}\sum_{j=1}^{3}(y_j - t_j)^2 = \tfrac{1}{2}((0.7-1)^2 + (0.3-0)^2 + (0.4-0)^2) = 0.17.$$

If, instead, the output is $\boldsymbol{y} = (0.3, 0.7, 0.4)$, then the sum-of-squares loss is:

$$L(\boldsymbol{y}, \boldsymbol{t}) = \tfrac{1}{2}\sum_{j=1}^{3}(y_j - t_j)^2 = \tfrac{1}{2}((0.3-1)^2 + (0.7-0)^2 + (0.4-0)^2) = 0.57.$$

Suppose that input $\boldsymbol{x}$ has been fed forward through a neural network, resulting in an output $\boldsymbol{y}$ and that the target for $\boldsymbol{x}$ is $\boldsymbol{t}$. Suppose, also, that the loss $L(\boldsymbol{y}, \boldsymbol{t})$ is non-zero. Then we want to change the weights in the network in such a way that if the same input $\boldsymbol{x}$ is fed forward through the changed network, to obtain a new output, then the loss between the new output and target is less than before. The method that we use to change, or **train**, the weights is gradient descent, which we discuss next.

2.9. **Gradient Descent Method.**

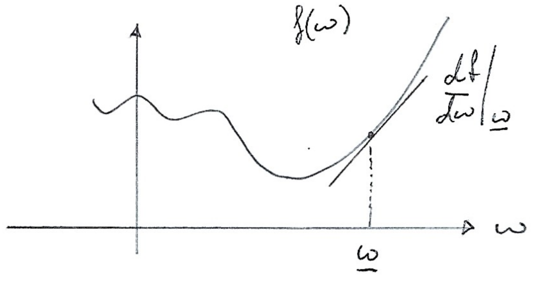
Let $f(w)$ be a differentiable function and let $\underline{w}$ be some point in the domain of f.
Then $f(\underline{w})$ is the value of f given input $\underline{w}$.
Suppose that we want change $\underline{w}$ in such a way that the value of $f(\underline{w})$ decreases slightly.
We use the gradient descent update rule, which is as follows:

$$\underline{w} \leftarrow \underline{w} - \eta \left(\frac{df}{dw}\Big|_{\underline{w}} \right),$$

where $\frac{df}{dw}\big|_{\underline{w}}$ is the derivative of f with respect to w, evaluated at $\underline{w}$, i.e., the gradient of f at $\underline{w}$. Here, η is the learning rate; we choose a small value for η so that the changes to $\underline{w}$ are in small increments.

Observe that if $\frac{df}{dw}\big|_{\underline{w}} > 0$, i.e., the gradient is positive, then $\underline{w}$ decreases, and if $\frac{df}{dw}\big|_{\underline{w}} < 0$ then $\underline{w}$ increases. Also, the change to $\underline{w}$ depends on the size of the gradient: the larger the gradient, the larger the change to $\underline{w}$.

After applying the gradient descent update rule to $\underline{w}$, the new value of $f(\underline{w})$ should be less than the original. If the above process is applied iteratively, then $f(\underline{w})$ should converge at a local minimum for f.



Next, let $F(w_1, \ldots, w_n)$ be a differentiable many-valued function and let $\underline{\boldsymbol{w}} = (\underline{w}_1, \ldots, \underline{w}_n)$ be some point in the domain of F. Then $F(\underline{\boldsymbol{w}})$ is the value of F given input $\underline{\boldsymbol{w}}$.
In this case, we want to change the values of all the $\underline{w}_i$s so that the value of $F(\underline{\boldsymbol{w}})$ decreases.

The method is very similar to the single variable case.

For each $\underline{w}_i$ apply the following gradient descent update rule:

$$\underline{w}_i \leftarrow \underline{w}_i - \eta \left(\frac{\partial F}{\partial w_i}\Big|_{\underline{\boldsymbol{w}}} \right),$$

where $\frac{\partial F}{\partial w_i}\Big|_{\underline{\boldsymbol{w}}}$ is the partial derivative of F with respect to w_i, evaluated at $\underline{\boldsymbol{w}}$.

After applying the gradient descent update rule to $\underline{w}$, the new value of $F(\underline{w})$ should be less than the original. If the above process is applied iteratively, then $F(\underline{w})$ should converge at a local minimum for F.

EXERCISES:

(1) Given the function $f(w) = \sigma(3w - 2)$ and the point $\underline{w} = 0.5$, apply the gradient descent update rule with $\eta = 0.1$.
(2) Given the function $F(w_1, w_2) = e^{(2w_1 - 3w_2 + 1)}$ and the point $\underline{\boldsymbol{w}} = (\underline{w}_1, \underline{w}_2) = (0.4, 0.5)$, apply the gradient descent update rule with $\eta = 0.1$.

2.10. **Chain rule for many-valued functions.**

We are going to apply the gradient descent method in neural networks. This will require the use of the **chain rule** for many-valued functions, so we recall the definition.

Let $F(z_1, \dots z_k)$ be a function in which each z_i is also a function. Let x be a variable that occurs in some of the z_i s, so F is also a function of x. Recall that $\frac{\partial F}{\partial x}$ is a function that gives the change to F resulting from a small change to the variable x. The chain rule allows the computation of $\frac{\partial F}{\partial x}$ in terms of the partial derivatives $\frac{\partial F}{\partial z_i}$, as follows:

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial z_1}\frac{\partial z_1}{\partial x} + \cdots + \frac{\partial F}{\partial z_k}\frac{\partial z_k}{\partial x} = \sum_{i=1}^{k} \frac{\partial F}{\partial z_i}\frac{\partial z_i}{\partial x}.$$

If x does not occur as a variable in some z_j, then $\frac{\partial z_j}{\partial x} = 0$.

Example: Suppose that $F(z_1, z_2, z_3) = z_1^2 - 3z_2 + e^{z_3}$ and $z_1 = xy$, $z_2 = x^2$ and $z_3 = y^3$. Then

$$\begin{aligned}
\tfrac{\partial F}{\partial x} &= \tfrac{\partial F}{\partial z_1}\tfrac{\partial z_1}{\partial x} + \tfrac{\partial F}{\partial z_2}\tfrac{\partial z_2}{\partial x} + \tfrac{\partial F}{\partial z_3}\tfrac{\partial z_3}{\partial x} \\
&= 2z_1 y - 3(2x) + e^{z_3}(0) \\
&= 2z_1 y - 6x, \\
\tfrac{\partial F}{\partial y} &= \tfrac{\partial F}{\partial z_1}\tfrac{\partial z_1}{\partial y} + \tfrac{\partial F}{\partial z_2}\tfrac{\partial z_2}{\partial y} + \tfrac{\partial F}{\partial z_3}\tfrac{\partial z_3}{\partial y} \\
&= 2z_1 x - 3(0) + e^{z_3} 3y^2 \\
&= 2z_1 x + e^{z_3} 3y^2.
\end{aligned}$$

2.11. **Gradient descent for neural networks.**

Assume that we have a neural network and an associated loss function L. It's important to note that the function L that we are trying to reduce has as its variables all the edge weights w and biases b in the network. We need to distinguish between the variables for the weights and biases and the current values of weights and biases. We use w and b (usually with subscripts) to denote variables, and $\underline{w}$ and $\underline{b}$ to denote the current values of these variables, as in the gradient descent method above.

We require some more notation. We use n, m and ℓ as names for nodes. If there is an edge from node m to node n then we say that m is *connected* to n and use w_{mn} as the variable for the weight of this edge. For each node n in the network, b_n denotes the variable for the bias at n (except at input nodes which don't have a bias). We use y_n as the variable for the activation value at n and z_n as the variable for the sum of inputs to the node n plus the bias. Thus,

$$z_n = \sum_{\ell} y_\ell w_{\ell n} + b_n,$$

where ℓ in the sum ranges over all nodes that are connected to n; these are nodes in the layer that precedes the layer where n is. Let g_n to denote n's activation function; then

$$y_n = g_n(z_n).$$

We show how to update the current values for all the $\underline{w}_{mn}$'s and also $\underline{b}_n$'s so as to reduce the loss.

Let $\boldsymbol{W}$ be the list of all current weights in the network, i.e., all $\underline{w}_{mn}$ and $\underline{b}_n$.

Suppose that $\boldsymbol{x}$ is an input to the network and the feedforward step with input $\boldsymbol{x}$ has been completed and output $\boldsymbol{y}$ has been obtained; note that the $\boldsymbol{y}$ values depend on the values in $\boldsymbol{x}$ and $\boldsymbol{W}$. Let $\boldsymbol{t}$ be the target corresponding to $\boldsymbol{x}$. Then $L(\boldsymbol{y}, \boldsymbol{t})$ is the loss corresponding to input $\boldsymbol{x}$. The gradient descent method is used to adjust the weights in the network so as to reduce the value of $L(\boldsymbol{y}, \boldsymbol{t})$ for the input $\boldsymbol{x}$.

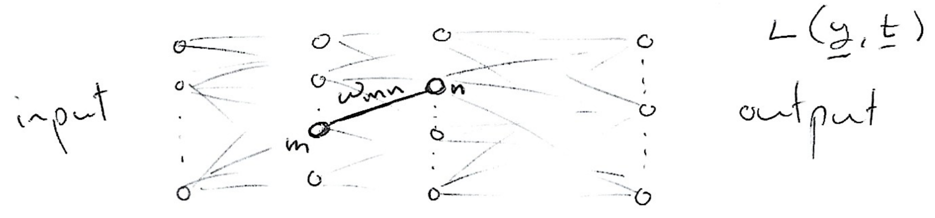
The values in $\boldsymbol{x}$ and $\boldsymbol{t}$ are considered constant at this point since they cannot be changed. The values in $\boldsymbol{W}$ are the current values of the weights at this point, which will be updated.

The notation a_n is used to denote the activation value at node n under the current weights and input, that is,

$$a_n = y_n\big|_{\boldsymbol{xtW}}.$$

With all the above notation, let us return to the gradient descent method.

Consider an edge weight $\underline{w}_{mn}$ in the given network, from node m to node n.



The gradient descent update rule for $\underline{w}_{mn}$ is:

$$\underline{w}_{mn} \leftarrow \underline{w}_{mn} - \eta \left(\frac{\partial L}{\partial w_{mn}}\Big|_{\boldsymbol{xtW}} \right). \tag{1}$$

To apply the above rule, we need to compute $\frac{\partial L}{\partial w_{mn}}\Big|_{\boldsymbol{xtW}}$.

The key to obtain a notationally simple version of gradient descent for neural networks is to work with the variables z_n (instead of y_n). We can do this since the y_n's are obtained directly from the z_n's by applying an activation function ($y_n = g_n(z_n)$).

Recall that the derivative $\frac{\partial L}{\partial w_{mn}}$ gives the change to L that results from to a small change to the variable w_{mn}. Since w_{mn} only contributes directly to the variable z_n, the chain rule can be applied as follows:

$$\frac{\partial L}{\partial w_{mn}} = \frac{\partial L}{\partial z_n} \frac{\partial z_n}{\partial w_{mn}}.$$

Since $z_n = \sum_\ell y_\ell w_{\ell n} + b_n$, where the ℓ in the sum ranges over all nodes that are connected to ℓ, one of which is m, we have that $\frac{\partial z_n}{\partial w_{mn}} = y_m$. Thus,

$$\frac{\partial L}{\partial w_{mn}} = \frac{\partial L}{\partial z_n}\, y_m.$$

Next, to compute $\frac{\partial L}{\partial w_{mn}}\Big|_{\boldsymbol{xtW}}$ we evaluate the above expression using the current values in $\boldsymbol{xtW}$:

(2) $$\frac{\partial L}{\partial w_{mn}}\Big|_{\boldsymbol{xtW}} = \left(\frac{\partial L}{\partial z_n}\, y_m\right)\Big|_{\boldsymbol{xtW}} = \left(\frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}\right)\left(y_m\Big|_{\boldsymbol{xtW}}\right) = \left(\frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}\right) a_m$$

where, in the last step, we use the notation $a_m = y_m\Big|_{\boldsymbol{xtW}}$.

We will also use the following notation.

For every node n, the **delta** value at n, denoted by δ_n, is defined as

$$\delta_n = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}.$$

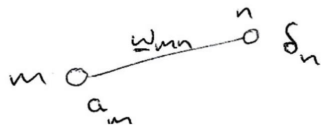
Using the delta notation, the expression in (2) can be written as:

$$\frac{\partial L}{\partial w_{mn}}\Big|_{\boldsymbol{xtW}} = \delta_n\, a_m.$$

Thus, the gradient descent update rule for $\underline{w}_{mn}$, given in (1), can be written as

(3) $$\underline{w}_{mn} \leftarrow \underline{w}_{mn} - \eta\, \delta_n\, a_m$$

The edge weight between **any** nodes m and n can be updated using (3) if δ_n and a_m are known.



Next, we consider the gradient descent update rule for a bias value. The gradient descent update rule for b_n is:

(4) $$\underline{b}_n \leftarrow \underline{b}_n - \eta\left(\frac{\partial L}{\partial b_n}\Big|_{\boldsymbol{xtW}}\right).$$

To apply the above rule, we need to compute $\frac{\partial L}{\partial b_n}\Big|_{\boldsymbol{xtW}}$.

The variable b_n only contributes directly to the variable z_n, hence we can apply the chain rule as follows:

$$\frac{\partial L}{\partial b_n} = \frac{\partial L}{\partial z_n}\frac{\partial z_n}{\partial b_n}.$$

Since $z_n = \sum_\ell y_\ell w_{\ell n} + b_n$, we have that $\frac{\partial z_n}{\partial b_n} = 1$. Thus,

$$\frac{\partial L}{\partial b_n} = \frac{\partial L}{\partial z_n}.$$

Then $\frac{\partial L}{\partial b_n}\Big|_{\boldsymbol{xtW}}$ is computed as follows:

$$\frac{\partial L}{\partial b_n}\Big|_{\boldsymbol{xtW}} = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}} = \delta_n.$$

Thus, the gradient descent update rule for $\underline{b}_n$, given in (4), can be written as

$$\underline{b}_n \leftarrow \underline{b}_n - \eta\,\delta_n. \tag{5}$$

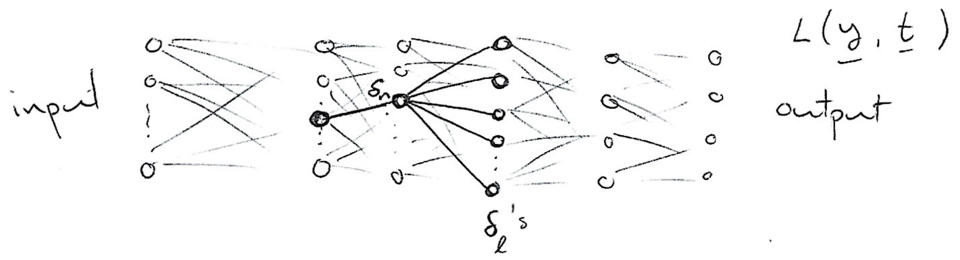
At **any** node n in the network, the bias value can be updated using (5) if δ_n is known.

2.12. Backpropagation.

In this section, we describe the backpropagation technique that uses the gradient descent method to train the edge weights and biases throughout a neural network. Backpropagation works by first obtaining the delta values at the nodes in the output layer and then propagating those values backwards through the network to obtain the delta values for all nodes and, thereby, updating all weights.

At the output nodes, the delta values (recall $\delta_n = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}$) are obtained directly from the chosen loss function L and the activation function g in the output layer. In subsequent sections we will consider some specific examples of loss functions and activation functions.

Let n be a node in the network. The value of δ_n can be obtained from the δ_ℓ values of the nodes ℓ in the layer that n is connected to.



The variable z_n contributes directly to the z_ℓ variables of the nodes in the layer subsequent to n's layer. Thus, the chain rule can be used as follows:

$$\frac{\partial L}{\partial z_n} = \sum_\ell \frac{\partial L}{\partial z_\ell}\frac{\partial z_\ell}{\partial z_n} \tag{6}$$

where the ℓ in the sum ranges over all nodes ℓ in the layer after n's layer.

For each node ℓ in the above sum, we will simplify $\frac{\partial z_\ell}{\partial z_n}$.
First, recall that $z_\ell = \sum_m y_m w_{m\ell} + b_\ell$, where m in the sum ranges over all nodes that are connected to node ℓ. Since n is one of the nodes connected to ℓ,

$$\frac{\partial z_\ell}{\partial y_n} = w_{n\ell}.$$

Then, by the chain rule,

$$\frac{\partial z_\ell}{\partial z_n} = \frac{\partial z_\ell}{\partial y_n}\frac{\partial y_n}{\partial z_n} = w_{n\ell}\frac{\partial y_n}{\partial z_n}.$$

We can write $\frac{dy_n}{dz_n}$ instead of $\frac{\partial y_n}{\partial z_n}$ since y_n is directly obtained from z_n. (Recall that $y_n = g_n(z_n)$; for this reason, $\frac{dy_n}{dz_n}$ is sometimes written as $g_n'(z_n)$.)

Thus, (6) simplifies to

$$\frac{\partial L}{\partial z_n} = \sum_\ell \frac{\partial L}{\partial z_\ell} w_{n\ell} \frac{dy_n}{dz_n} = \left(\sum_\ell \frac{\partial L}{\partial z_\ell} w_{n\ell}\right)\frac{dy_n}{dz_n}.$$

Finally,

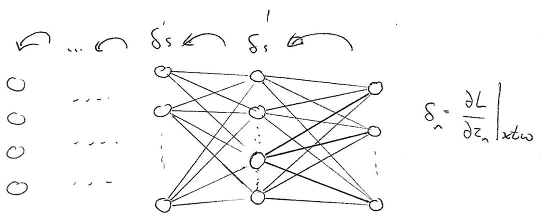
$$\begin{aligned}
\delta_n = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}} &= \left(\sum_\ell \left(\frac{\partial L}{\partial z_\ell} w_{n\ell}\right)\Big|_{\boldsymbol{xtW}}\right)\left(\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right) \\
&= \left(\sum_\ell \frac{\partial L}{\partial z_\ell}\Big|_{\boldsymbol{xtW}} w_{n\ell}\Big|_{\boldsymbol{xtW}}\right)\left(\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right) \\
&= \left(\sum_\ell \delta_\ell\, \underline{w}_{n\ell}\right)\left(\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right).
\end{aligned}$$

To summarise,

$$\delta_n = \left(\sum_\ell \delta_\ell\, \underline{w}_{n\ell}\right)\left(\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right) \tag{7}$$

where the ℓ in the sum ranges over all nodes ℓ in the layer after n's layer.

Once the delta values at the ouput layer are calculated, which depends on the chosen loss function and activation functions, (7) is used to compute the delta values at the layer preceding the output layer. These values, in turn are used to compute delta values at the preceding layer, and so on.



2.13. **Pseudocode for training neural networks.**

Neural Network Training Algorithm

```
Accepts a dataset containing an array of input values and an array of target values.

Create a neural network with the required number of layers
and the required number of nodes in each layer.
Create matrices for the edge weights between each pair of successive layers
and fill the matrices with random values.
Create arrays for the bias values for each layer and fill with random values.
Choose an activation function for each layer.
(Usually the random values are chosen from a fixed range such as $[-1, 1]$.)
(We use $\boldsymbol{W}$ to denote the current values of edge weights and biases.)

Choose a learning rate $\eta$   (usually between 0.001 and 0.1).

while (stopping condition is not satisfied)
  for each input $\boldsymbol{x}$ with target $\boldsymbol{t}$, do the following:
      feedforward input $\boldsymbol{x}$ to get output $\boldsymbol{y}$
          store output values and activation values at hidden layers (the $a_m$'s)

      for each output node $n$, compute $\delta_n = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}$
      for every node $m$ in the preceding layer, compute $\delta_m = \left(\sum_\ell \delta_\ell \underline{w}_{m\ell}\right)\left(\frac{dy_m}{dz_m}\Big|_{\boldsymbol{xtW}}\right)$,
        where $\ell$ ranges over all nodes in the output layer

      for every $m$ in layer preceding the output layer and every $n$ in the output layer,
          update weight $\underline{w}_{mn}$ using:   $\underline{w}_{mn} \leftarrow \underline{w}_{mn} - \eta\, \delta_n\, a_m$
          update bias $\underline{b}_n$ using:   $\underline{b}_n \leftarrow \underline{b}_n - \eta\, \delta_n$

      continue in this way with each preceding layer until the input layer.

      for every node $\ell$ in the input layer and every node $m$ in the next layer,
          update weight $\underline{w}_{\ell m}$ using:   $\underline{w}_{\ell m} \leftarrow \underline{w}_{\ell m} - \eta\, \delta_m\, x_\ell$
          update bias $\underline{b}_m$ using:   $\underline{b}_m \leftarrow \underline{b}_m - \eta\, \delta_m$
```

2.14. **Using sum-of-squares loss function.**

We consider a particular case of the Neural Network Training Algorithm in which the sum-of-squares loss function is used.
Note that the choice of loss function only affects the output layer and, in particular, the calculation of δ_n values at the output nodes.

As before, let $\boldsymbol{W}$ be the list of all current weights in the network. Suppose that $\boldsymbol{x}$ is an input to the network and the feedforward step with input $\boldsymbol{x}$ has been completed and output $\boldsymbol{y}$ has been obtained. Also, let $\boldsymbol{t}$ be the target value corresponding to $\boldsymbol{x}$.

The sum-of-squares loss is given by the following function, where the sum ranges over all output nodes ℓ:

$$L(\boldsymbol{y},\boldsymbol{t}) = \tfrac{1}{2}\sum_{\ell}(y_\ell - t_\ell)^2.$$

Consider a particular output node, say n.
The following derivation is used to calculate $\delta_n = \frac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}}$:

$$\begin{aligned}
\tfrac{\partial L}{\partial z_n} &= \tfrac{\partial}{\partial z_n}\tfrac{1}{2}\sum_{\ell}(y_\ell - t_\ell)^2 \\
&= \tfrac{1}{2}\sum_{\ell}\tfrac{\partial}{\partial z_n}(y_\ell - t_\ell)^2 \\
&= \tfrac{1}{2}\sum_{\ell}2(y_\ell - t_\ell)(\tfrac{\partial y_\ell}{\partial z_n} - \tfrac{\partial t_\ell}{\partial z_n}) \\
&= \sum_{\ell}(y_\ell - t_\ell)\tfrac{\partial y_\ell}{\partial z_n} \quad (\text{since each } \tfrac{\partial t_\ell}{\partial z_n} = 0) \\
&= (y_n - t_n)\tfrac{\partial y_n}{\partial z_n} \quad (\text{since } \tfrac{\partial y_\ell}{\partial z_n} = 0 \text{ if } \ell \text{ and } n \text{ are different}).
\end{aligned}$$

Recall that we can write $\frac{dy_n}{dz_n}$ instead of $\frac{\partial y_n}{\partial z_n}$ since $y_n = g_n(z_n)$.
Thus, we have that:

$$\tfrac{\partial L}{\partial z_n}\Big|_{\boldsymbol{xtW}} = \left((y_n - t_n)\tfrac{dy_n}{dz_n}\right)\Big|_{\boldsymbol{xtW}} = (a_n - t_n)\left(\tfrac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right)$$

and, therefore,

$$\delta_n = (a_n - t_n)\left(\tfrac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right).$$

The next step in computing δ_n is to evaluate $\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}$, and this depends on the choice of activation function g_n.

2.15. **Using sum-of-squares loss function and sigmoid activation function.**

We consider a particular case of the Neural Network Training Algorithm in which the activation function used at every node is sigmoid and the sum-of-squares loss function is used.

We need to calculate the values δ_n at the output nodes.

From the previous section, we have that

$$\delta_n = (a_n - t_n)\left(\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}\right)$$

so we need to calculate $\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}}$ in the case that g_n is σ, that is, $y_n = \sigma(z_n)$.

In the exercises, you will show that if $y_n = \sigma(z_n)$, then $\frac{dy_n}{dz_n} = \sigma(z_n)(1 - \sigma(z_n))$,
and hence that $\frac{dy_n}{dz_n} = y_n(1 - y_n)$.

Then, recalling that $y_n\Big|_{\boldsymbol{xtW}} = a_n$, we have

$$\frac{dy_n}{dz_n}\Big|_{\boldsymbol{xtW}} = y_n(1 - y_n)\Big|_{\boldsymbol{xtW}} = a_n(1 - a_n).$$

Finally, the formula for computing δ_n at the output nodes is:

$$\delta_n = (a_n - t_n)a_n(1 - a_n). \tag{8}$$

Consider a node m in a hidden layer. The calculation of δ_m is given by:

$$\delta_m = \left(\textstyle\sum_n \delta_n \, \underline{w}_{mn}\right)\left(\frac{dy_m}{dz_m}\Big|_{\boldsymbol{xtW}}\right)$$

where the n's range over the nodes in the layer after m's layer.

Since we are using σ as the activation function at the hidden layer, we have that $y_m = \sigma(z_m)$.
As above, $\frac{dy_m}{dz_m} = y_m(1 - y_m)$, so

$$\frac{dy_m}{dz_m}\Big|_{\boldsymbol{xtW}} = y_m(1 - y_m)\Big|_{\boldsymbol{xtW}} = a_m(1 - a_m).$$

Thus,

$$\delta_m = \left(\textstyle\sum_n \delta_n \, \underline{w}_{mn}\right) a_m(1 - a_m) \tag{9}$$

where n ranges over all nodes in the layer after m's layer.

The formulas obtained in (8) and (9) can now be used in the Neural Network Training Algorithm. We give the pseudocode below in the case of a 3-layer network (i.e., with one hidden layer).

Neural Network Training Algorithm

for 3-layer network with σ activations and sum-of-squares loss

```
Accepts a dataset containing an array of input values and an array of target values.

Create a neural network with input later, one hidden layer and output layer
with the required number of nodes in each layer as follows.
Create matrices for the edge weights between each pair of successive layers
and fill the matrices with random values.
Create arrays for the bias values for each layer and fill with random values.
Set $\sigma$ as activation function for each layer.
(Usually the random values are chosen from a fixed range such as $[-1, 1]$.)
(We use $\boldsymbol{W}$ to denote the combination of all current edge weights and biases.)

Choose a learning rate $\eta$   (usually between 0.001 and 0.1).

while (stopping condition is not satisfied)
  for each input $\boldsymbol{x}$ with target $\boldsymbol{t}$, do the following:
      feedforward input $\boldsymbol{x}$ to get output $\boldsymbol{y}$
          store the output values and also the activation values at the hidden layer

      for each output node $n$, where $a_n$ is the output value, compute $\delta_n = (a_n - t_n)a_n(1 - a_n)$
      for every node $m$ in the hidden layer, where $a_m$ is the activation value,
          compute $\delta_m = \left(\sum_n \delta_n \underline{w}_{mn}\right) a_m(1 - a_m)$,
          where $n$ ranges over all output nodes

      for every node $m$ in the hidden layer and node $n$ in the output layer,
          update weight $\underline{w}_{mn}$ using:   $\underline{w}_{mn} \leftarrow \underline{w}_{mn} - \eta\, \delta_n\, a_m$
          update bias $\underline{b}_n$ using:   $\underline{b}_n \leftarrow \underline{b}_n - \eta\, \delta_n$

      for every node $\ell$ in the input layer and every node $m$ in the hidden layer,
          update weight $\underline{w}_{\ell m}$ using:   $\underline{w}_{\ell m} \leftarrow \underline{w}_{\ell m} - \eta\, \delta_m\, x_\ell$
          update bias $\underline{b}_m$ using:   $\underline{b}_m \leftarrow \underline{b}_m - \eta\, \delta_m$
```

EXERCISES:

(1) If $y = \sigma(z)$, show that $\frac{dy}{dz} = \sigma(z)(1 - \sigma(z)) = y(1 - y)$.

(2) Consider a network with 2 input nodes, one hidden layer with 2 nodes, and 2 output nodes. The weights and the bias values are given by W_1, W_2, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$:

$$W_1 = \begin{bmatrix} 1 & -1 \\ -3 & -2 \end{bmatrix} \quad W_2 = \begin{bmatrix} -2 & 0 \\ 0 & 3 \end{bmatrix} \quad \boldsymbol{b}_1 = (0, 1) \quad \boldsymbol{b}_2 = (1, -2).$$

The activation function at each node is the sigmoid function.

Suppose we want to train the network using input $\boldsymbol{x} = (1, -2)$ with target $\boldsymbol{t} = (1, 0)$.

(a) Calculate the output vector $\boldsymbol{y} = (y_1, y_2)$ for the input $\boldsymbol{x}$.

(b) Compute the sum-of-squares loss for the input $\boldsymbol{x}$.

(c) Do the weight updates for each edge weight and bias according to the NEURAL NETWORK TRAINING ALGORITHM with sum-of-squares loss and sigmoid activation functions.

(d) After the updates, feed $\boldsymbol{x}$ into the updated network to get the output.

(e) Compute the sum-of-squares loss for the input $\boldsymbol{x}$ and compare with (b) (it should have decreased).

(3) Consider a neural network that uses the sum-of-squares loss function and the *relu* activation function at each layer. Do the computation in Section 2.15 to obtain the rules for computing delta values, that is, obtain the formulas (8) and (9) in the case of the *relu* activation function.

(4) Try exercise (3) with the activation function *lin*.

(5) Try exercise (3) with the activation function *tanh*.

You need to show that if $y = tanh(z)$, then $\frac{dy}{dz} = 1 - tanh^2(z) = 1 - y^2$.