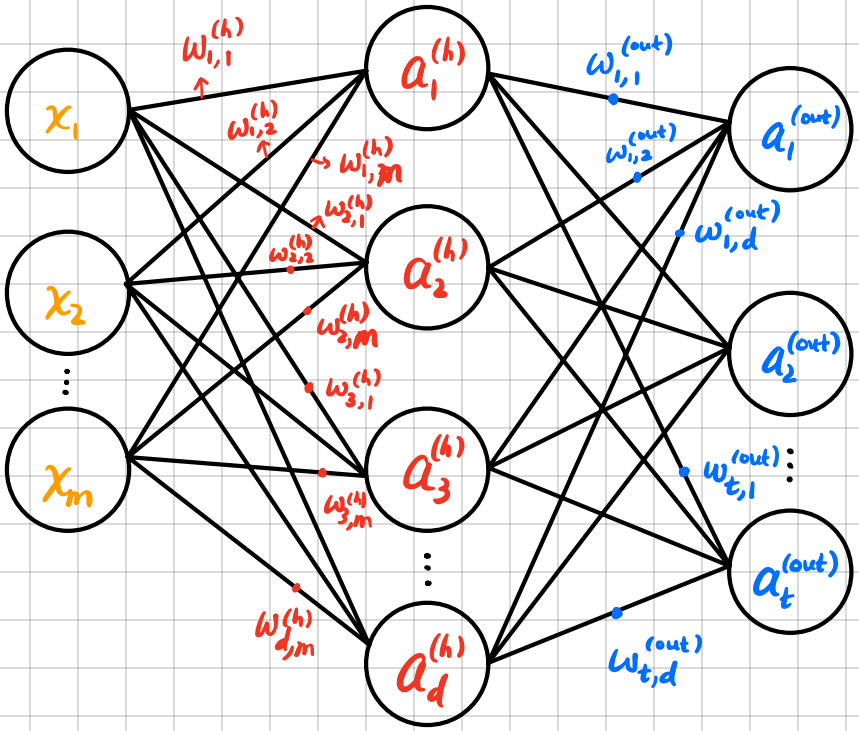
Multilayer perceptron (MLP)

- Deep learning
- Deep neural network
- Modeling complex functions

Consider the simplest MLP which consists of only one hidden layer



input layer — m neurons

hidden layer — d neurons

output layer — t neurons

- The number of neurons in the input layer is the number of features in the dataset, denoted as m.
- The number of neurons in the output layer is the number of labels or classes in the dataset, denoted as t.

- The number of neurons in the hidden layer is denoted as d.

  It is a hyperparameter.

- The weights between input layer and hidden layer is denoted as

$$w_{j,k}^{(h)}$$

  which connects the j-th neuron of the hidden layer and the k-th neuron of the input layer. The superscript (h) indicates that it's the weight between input and hidden layers.

- Similarly, the weights between hidden layer and output layer is denoted as

$$w_{j,k}^{(out)}$$

  which connects the j-th neuron of the hidden layer and the k-th neuron of the input layer. The superscript (out) indicates that it's the weight between input and hidden layers.

- The neuron value of the hidden layer can be obtained by first calculating the weighted sum

$$z_1^{(h)} = x_1 w_{1,1}^{(h)} + x_2 w_{1,2}^{(h)} + x_3 w_{1,3}^{(h)} + \cdots + x_m w_{1,m}^{(h)} + b_1^{(h)}$$

$$z_2^{(h)} = x_1 w_{2,1}^{(h)} + x_2 w_{2,2}^{(h)} + x_3 w_{2,3}^{(h)} + \cdots + x_m w_{2,m}^{(h)} + b_2^{(h)}$$

$$\vdots$$

$$z_d^{(h)} = x_1 w_{d,1}^{(h)} + x_2 w_{d,2}^{(h)} + x_3 w_{d,3}^{(h)} + \cdots + x_m w_{d,m}^{(h)} + b_d^{(h)}$$

  and then apply the nonlinear activation function

$$a_1^{(h)} = \sigma(z_1^{(h)})$$

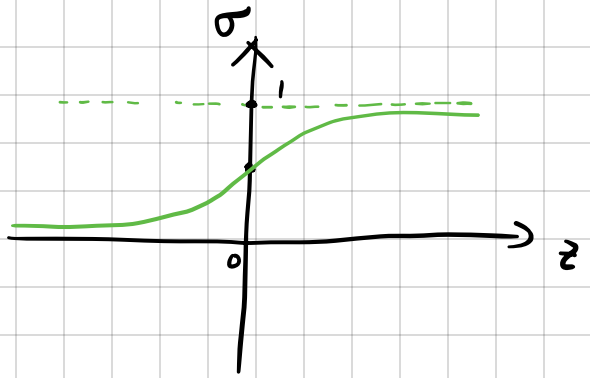
$$a_2^{(h)} = \sigma(z_2^{(h)})$$

$$\vdots$$

$$a_d^{(h)} = \sigma(z_d^{(h)})$$

where we have introduced $b_1^{(h)}, b_2^{(h)}, \cdots, b_d^{(h)}$ as the bias

and $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.



For the purpose of code efficiency and readability, the above equations for individual neurons in the hidden layer can be reformulated as

Note: we need to take the transpose

$$\vec{z}^{(h)} = \vec{x}\, W^{(h)T} + \vec{b}^{(h)}$$

$$\vec{a}^{(h)} = \sigma(\vec{z}^{(h)})$$

where the row vectors are defined as

dimension

$$\vec{z}^{(h)} = [z_1^{(h)}, z_2^{(h)}, z_3^{(h)}, \cdots, z_d^{(h)}] \quad (1 \times d)$$

$$\vec{a}^{(h)} = [a_1^{(h)}, a_2^{(h)}, a_3^{(h)}, \cdots, a_d^{(h)}] \quad (1 \times d)$$

$$\vec{b}^{(h)} = [b_1^{(h)}, b_2^{(h)}, b_3^{(h)}, \cdots, b_d^{(h)}] \quad (1 \times d)$$

$$\vec{x} = [x_1, x_2, \cdots, x_m] \qquad (1 \times m)$$

and the weight matrix

$$W^{(h)} = \begin{bmatrix} w_{1,1}^{(h)} & w_{1,2}^{(h)} & \cdots & w_{1,m}^{(h)} \\ w_{2,1}^{(h)} & w_{2,2}^{(h)} & \cdots & w_{2,m}^{(h)} \\ w_{3,1}^{(h)} & w_{3,2}^{(h)} & \cdots & w_{3,m}^{(h)} \\ \vdots & \vdots & & \vdots \\ w_{d,1}^{(h)} & w_{d,2}^{(h)} & \cdots & w_{d,m}^{(h)} \end{bmatrix} \qquad (d \times m)$$

Its transpose

$$W^{(h)T} = \begin{bmatrix} w_{1,1}^{(h)} & w_{2,1}^{(h)} & \cdots & w_{d,1}^{(h)} \\ w_{1,2}^{(h)} & w_{2,2}^{(h)} & \cdots & w_{d,2}^{(h)} \\ \vdots & \vdots & & \vdots \\ w_{1,m}^{(h)} & w_{2,m}^{(h)} & \cdots & w_{d,m}^{(h)} \end{bmatrix} \qquad (m \times d)$$

- The above equations are the forward calculations from input layer to the hidden layer. Similarly, the forward calculations from the hidden layer to the output layer would be

(the hidden layer neuron values become the input)

$$\underset{(1\times t)}{\vec{z}^{(out)}} = \underset{(1\times d)}{\vec{a}^{(h)}} \; \underset{(t\times d)^T}{W^{(out)T}} + \underset{(1\times t)}{\vec{b}^{(out)}}$$

$$\underset{(1\times t)}{\vec{a}^{(out)}} = \sigma\left(\underset{(1\times t)}{\vec{z}^{(out)}}\right)$$

t: # of neurons in output layer.

- The above are the forward calculations for a single sample.

The input data usually contains n samples, which form a matrix

$$X^{(in)} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & x_3^{[1]} & \cdots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & x_3^{[2]} & \cdots & x_m^{[2]} \\ x_1^{[3]} & x_2^{[3]} & x_3^{[3]} & \cdots & x_m^{[3]} \\ \vdots & \vdots & \vdots & & \vdots \\ x_1^{[n]} & x_2^{[n]} & x_3^{[n]} & \cdots & x_m^{[n]} \end{bmatrix}$$

← 1st sample
← 2nd sample
← 3rd sample
← n-th sample

sample index

feature index

1st feature
2nd feature
3rd feature
m-th feature

Each sample, when fed into the neural network, generates

$$\underset{(1\times d)}{\vec{z}^{(1)}} \rightarrow \underset{(1\times d)}{\vec{a}^{(h)}} \rightarrow \underset{(1\times t)}{\vec{z}^{(out)}} \rightarrow \underset{(1\times t)}{\vec{a}^{(out)}}$$

Thus, they all can be extended to a matrix to include the new sample dimension.

$$\underset{(n\times d)}{Z^{(h)}} \rightarrow \underset{(n\times d)}{A^{(h)}} \rightarrow \underset{(n\times t)}{Z^{(out)}} \rightarrow \underset{(n\times t)}{A^{(out)}}$$

Explicitly, we have

$$Z^{(h)} = \begin{bmatrix} z_1^{[1]} & z_2^{[1]} & z_3^{[1]} & \cdots & z_d^{[1]} \\ z_1^{[2]} & z_2^{[2]} & z_3^{[2]} & \cdots & z_d^{[2]} \\ \vdots & \vdots & \vdots & & \vdots \\ z_1^{[n]} & z_2^{[n]} & z_3^{[n]} & & z_d^{[n]} \end{bmatrix}$$

sample index

hidden layer neuron index

- The relation between these matrices are

$$\underset{(n\times d)}{Z^{(h)}} = \underset{(n\times m)}{X^{(in)}}\ \underset{(d\times m)^T}{W^{(h)T}} + \underset{(1\times d)}{\vec{b}^{(h)}}$$

$$\underset{(n\times d)}{A^{(h)}} = \sigma\left(\underset{(n\times d)}{Z^{(h)}}\right)$$

$$\underset{(n\times t)}{Z^{(out)}} = \underset{(n\times d)}{A^{(h)}}\ \underset{(t\times d)^T}{W^{(out)T}} + \underset{(1\times t)}{\vec{b}^{(out)}}$$

$$\underset{(n\times t)}{A^{(out)}} = \sigma\left(\underset{(n\times t)}{Z^{(out)}}\right)$$

Note: $\vec{b}$ is a row vector for hidden layer and output layer. When added to the product of $X^{(in)}$ and $W^{(h)T}$ or $A^{(h)}$ and $W^{(out)T}$, which is a matrix, the $\vec{b}$ vector is added to each row of that matrix. or equivalently, $\vec{b}^{(h)}$ and $\vec{b}^{(out)}$ should be of the following form

$$\begin{bmatrix} b_1^{(h)} & b_2^{(h)} & b_3^{(h)} & \cdots & b_d^{(h)} \\ b_1^{(h)} & b_2^{(h)} & b_3^{(h)} & \cdots & b_d^{(h)} \\ \vdots & \vdots & \vdots & & \vdots \\ b_1^{(h)} & b_2^{(h)} & b_3^{(h)} & & b_d^{(h)} \end{bmatrix} \quad (n \times d)$$

and

$$\begin{bmatrix} b_1^{(out)} & b_2^{(out)} & b_3^{(out)} & \cdots & b_t^{(out)} \\ b_2^{(out)} & b_2^{(out)} & b_3^{(out)} & \cdots & b_t^{(out)} \\ \vdots & \vdots & \vdots & & \vdots \\ b_1^{(out)} & b_2^{(out)} & b_3^{(out)} & \cdots & b_t^{(out)} \end{bmatrix} \quad (n \times t)$$

- The nonlinear activation function acts on each element of the corresponding matrix. $\Rightarrow$ It's an element-wise operation.

Backpropagation

- After the forward calculation, we can compare the output to the actual label and introduce the following loss function

$$L\left(W^{(h)}, \vec{b}^{(h)}, W^{(out)}, \vec{b}^{(out)}\right) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{t} \sum_{j=1}^{t} \left(y_j^{[i]} - a_j^{(out)[i]} \right)^2$$

actual label; sample index

n: total # of samples; t: total # of neurons; j: output layer neuron index

- Our goal is to minimize this loss function to find the optimal weights and biases. In total, the number of parameters is

$$d \times m + d + t \times d + t = d(m+1) + t(d+1).$$

The loss function could be very complex, and nonconvex in this high-dimensional parameter space.

- Each of these parameters shall be optimized using gradient descent. For instance, one of the weights, $w_{j,\ell}^{(h)}$, which connects the j-th hidden layer and the ℓ-th neuron of the input layer, should be updated as

new weight — old weight — learning rate

$$\boxed{w_{j,\ell}^{(h)} := w_{j,\ell}^{(h)} - \eta \frac{\partial L}{\partial w_{j,\ell}^{(h)}}}$$

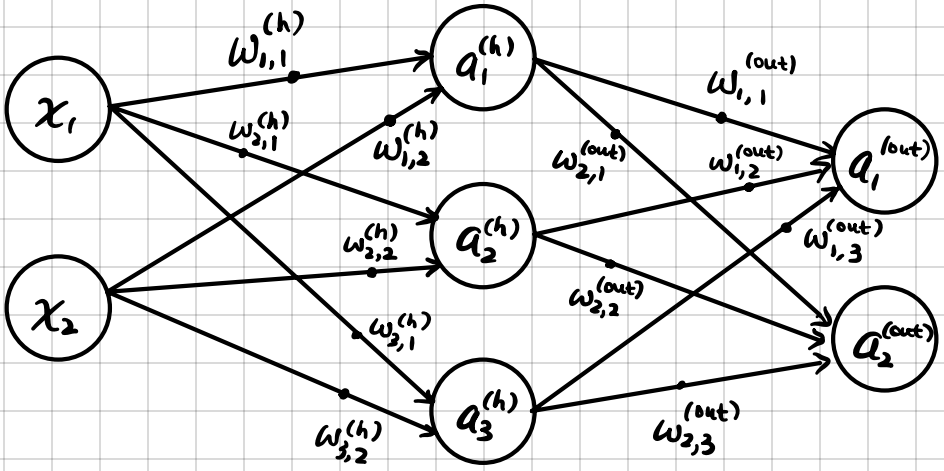
and one of the biases, say $b_k^{(out)}$, which is the bias for the k-th nuron of the output layer, should be updated as

$$\boxed{b_k^{(out)} := b_k^{(out)} - \eta \frac{\partial L}{\partial b_k^{(out)}}}$$

- The main task is to find these derivatives, which can be computed using the backpropagation algorithm. The key step for derivation is the chain rule. For a nested function $f(g(x))$, we have

$$\frac{d}{dx}[f(g(x))] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

- To demonstate the backpropagation algorithm, we consider a simple network



There are only 2, 3, and 2 neurons in the input, hidden, output layers.

Also, for now, assume there is only one sample in the dataset.

- According to previous discussions,

$$z_1^{(h)} = x_1 w_{1,1}^{(h)} + x_2 w_{1,2}^{(h)} + b_1^{(h)}$$

$$a_1^{(h)} = \sigma(z_1^{(h)})$$

$$\vdots$$

$$z_1^{(out)} = a_1^{(h)} w_{1,1}^{(out)} + a_2^{(h)} w_{1,2}^{(out)} + a_3^{(h)} w_{1,3}^{(out)} + b_1^{(out)}$$

$$a_1^{(out)} = \sigma(z_1^{(out)})$$

$$\vdots$$

and

$$L = \frac{1}{t} \sum_{j=1}^{t=2} \left(y_j - a_j^{(out)} \right)^2$$

(y_j: actual label; $a_j^{(out)}$: output neuron value)

- First, let's compute the derivative

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \boxed{\frac{\partial L}{\partial a_1^{(out)}} \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}}} \frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}}$$

expression in the box

$\delta_1^{(out)}$: in the following, we'll introduce a new notation for the

Since $\frac{\partial L}{\partial a_1^{(out)}} = (y_1 - a_1^{(out)}) \cdot (-1) = a_1^{(out)} - y_1$

$$\frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = -(1+e^{-z})^{-2} e^{-z} \cdot (-1)$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \frac{e^{-z}}{1+e^{-z}} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

Note: in the middle steps, we've droped the superscript (out)

$$= a_1^{(out)}\left(1 - a_1^{(out)}\right)$$

$$\frac{\partial z_1^{(out)}}{\partial w_{1,1}^{(out)}} = a_1^{(h)}$$

Putting all of it together,

$$\frac{\partial L}{\partial w_{1,1}^{(out)}} = \left(a_1^{(out)} - y_1\right) a_1^{(out)} \left(1 - a_1^{(out)}\right) a_1^{(h)}$$

Note: The new notation introduced above will be reused to compute the derivatives w.r.t. the hidden layer's weights.

$$\boxed{\delta_1^{(out)} = \frac{\partial L}{\partial a_1^{(out)}} \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}} = \left(a_1^{(out)} - y_1\right) a_1^{(out)} \left(1 - a_1^{(out)}\right)}$$

For example: to compute $\frac{\partial L}{\partial w_{1,1}^{(h)}}$, we need to know all the following quantities that depend on $w_{1,1}^{(h)}$ and then apply the chain rule.

Based on our forward calculation, $w_{1,1}^{(h)}$ is used to compute $a_1^{(h)}$ in the hidden layer and then $a_1^{(out)}$ and $a_2^{(out)}$ in the output layer.

Hence, mathematically, we have

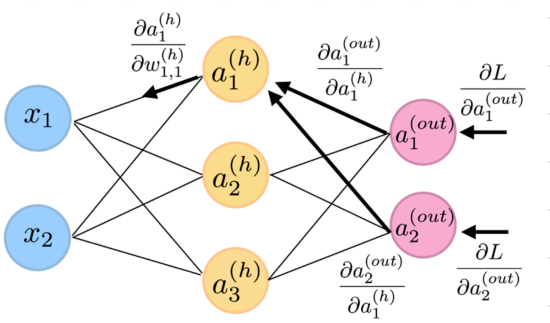
this is just $\delta_1^{(out)}$ introduced on the previous page

$$\frac{\partial L}{\partial w_{1,1}^{(h)}} = \boxed{\frac{\partial L}{\partial a_1^{(out)}} \frac{\partial a_1^{(out)}}{\partial z_1^{(out)}}} \frac{\partial z_1^{(out)}}{\partial a_1^{(h)}} \frac{\partial a_1^{(h)}}{\partial z_1^{(h)}} \frac{\partial z_1^{(h)}}{\partial w_{1,1}^{(h)}} + \boxed{\frac{\partial L}{\partial a_2^{(out)}} \frac{\partial a_2^{(out)}}{\partial z_2^{(out)}}} \frac{\partial z_2^{(out)}}{\partial a_1^{(h)}} \frac{\partial a_1^{(h)}}{\partial z_1^{(h)}} \frac{\partial z_1^{(h)}}{\partial w_{1,1}^{(h)}}$$

($\frac{\partial z_1^{(out)}}{\partial a_1^{(h)}}$ uses $w_{1,1}^{(out)}$; $\frac{\partial z_2^{(out)}}{\partial a_1^{(h)}}$ uses $w_{2,1}^{(out)}$; the last two factors are the same: $a_1^{(h)}(1-a_1^{(h)})x_1$)

Similarly, this can be denoted as $\delta_2^{(out)}$ which has a similar expression as $\delta_1^{(out)}$

$$= \boxed{\delta_1^{(out)}} w_{1,1}^{(out)} a_1^{(h)}(1-a_1^{(h)})x_1 + \boxed{\delta_2^{(out)}} w_{2,1}^{(out)} a_1^{(h)}(1-a_1^{(h)})x_1$$



Hence, if $\delta_1^{(out)}$ and $\delta_2^{(out)}$ are known (as calculated previously), then the derivative $\frac{\partial L}{\partial w_{1,1}^{(h)}}$ can be obtained straightforwardly.