Term explanation

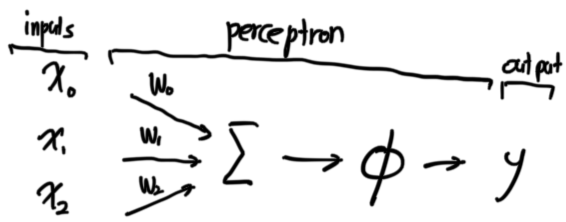
P^l_n : n-th perceptron of P^l_n-th layer or the output value of it.

W^L_{nm} : the weight about m-th input value of P^l_n.

x_{nm} : m-th input of n-th perceptron in 0-th layer.

How does a single perceptron calculate :

There's only one perception so we can omit the perceptron number and the layer number.



The Σ and the ϕ of the above image are functions.

The Σ calculates $\sum_{i=0}^{N-1} x_i w_i$ when N is the number of the inputs.

The ϕ is a function called 'activation function' which makes linear function to non-linear.

Sigmoid function ($y=\frac{1}{1+e^{-x}}$) and ReLU ($y=\begin{cases} x & (x>0) \\ 0 & (x<0)\end{cases}$) are the examples of the activation function

So we can write the formula of above perceptron like this :

$$\phi(\Sigma(w,x)) = \phi\left(\sum_{i=0}^{N-1} w_i x_i\right)$$

(w and x are sequences of $\underbrace{\text{weights}}_{(w_0, w_1, \cdots, w_{N-1})}$ and $\underbrace{\text{inputs}}_{(x_0, x_1, \cdots, x_{N-1})}$)

How does a single perceptron learn

Consider following case :

The perceptron have to print $\hat{y}$ as the output when we give 2 specific number x_0 and x_1 as the input.

And the real output of the perceptron is y.

There is difference between $\hat{y}$ (the value we expect) and y (the value of perceptron's calculation).

We can evaluate this difference by cost function (J) which calculate the degree of error.

So $J(y,\hat{y})$ is the degree of error between y and $\hat{y}$.

Then we can adjust the specific weight w_k like this:

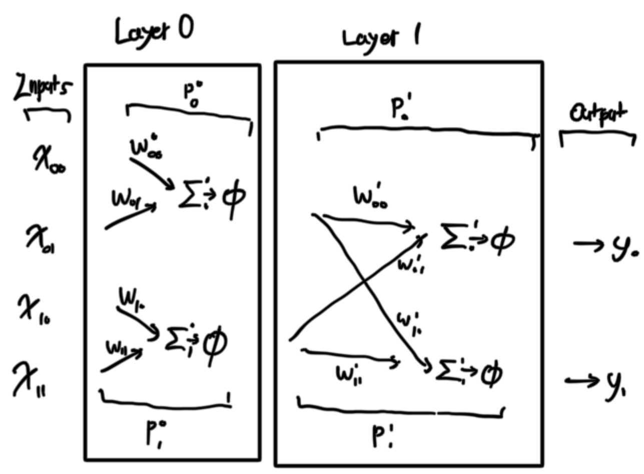
$$\underbrace{w_k}_{\text{adjusted weight}} \leftarrow \underbrace{w_k}_{\text{old weight}} - \frac{J(y,\hat{y})}{dw_k}$$

* This is from 'gradient descent', the algorithm for adjusting the weights. I omitted the expanation of this because this isn't main subject.

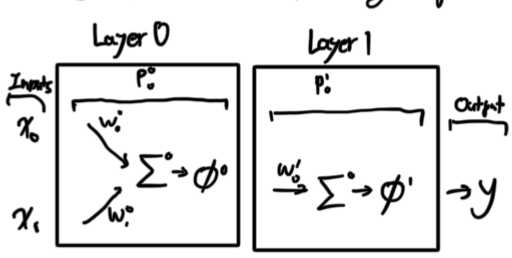
The above process is what we call 'learning of perceptron'.

How does multi layer perceptron learn

Multi layer perceptron (model) looks like this:



Or we can think about following simple one:



P^0_0 calculates like this:

$$P^0_0 = \phi^1\left(\sum_{i=0}^{1} x_i w^0_i\right)$$

P^1_0 calculates like this:

$$P^1_0 = \phi^1\left(\sum_{i=0}^{0} P^0_i w^1_i\right)$$

And the whole perceptron model calculates like this:

$$y = \phi^1\left(\sum_{i=0}^{0} \left(\phi^0\left(\sum_{j=0}^{1} x_j w^0_j\right) w^1_i\right)\right)$$

And the error level is like this:

$$J(y, \hat{y})$$

We have to get the value of $\frac{dJ(y,\hat{y})}{dw^1_0}$, $\frac{dJ(P^0_0,\hat{P}^0_0)}{dw^0_0}$, $\frac{dJ(P^0_1,\hat{P}^0_1)}{dw^0_1}$ to train the model.

We can easily get $\frac{dJ(y,\hat{y})}{dw^1_0}$ but how can we get $\frac{dJ(P^0_0,\hat{P}^0_0)}{dw^0_0}$ and $\frac{dJ(P^0_1,\hat{P}^0_1)}{dw^0_1}$ when we don't know $\hat{P}^0_0$ and $\hat{P}^0_1$, which mean the ideal value of P^0_0 and P^0_1?

To do this, $\frac{dJ(y,\hat{y})}{dw_0^0}$ and $\frac{dJ(y,\hat{y})}{dw_i}$ can be used instead of them.

Let me calculate $\frac{dJ(y,\hat{y})}{dw_i}$ as an example.

$$\frac{dJ(y,\hat{y})}{dw_i} = \frac{d}{dw_i^0} J\left(\phi^1\left(\sum_{i=0}^{0}\left(\phi^0\left(\sum_{j=0}^{1} x_j w_j^0\right) w_i^1\right), \hat{y}\right) = \frac{dJ}{d\phi^1} \cdot \frac{d}{dw_0^1} \phi^1\left(\sum_{i=0}^{0}\left(\phi^0\left(\sum_{j=0}^{1} x_j w_j^0\right) w_i^1\right)\right.$$

$$= \frac{dJ}{d\phi^1} \cdot \frac{d\phi^1}{d\Sigma^1} \cdot \frac{d}{dw_0^1} \sum_{i=0}^{0}\left(\phi^0\left(\sum_{j=0}^{1} x_j w_j^0\right) w_i^1\right) = \cdots$$

$$= \frac{dJ}{d\phi^1} \cdot \frac{d\phi^1}{d\Sigma^1} \cdot \frac{d\Sigma^1}{d\phi^0} \cdot \frac{d\phi^0}{d\Sigma^0} \cdot \frac{d\Sigma^0}{dw_i^0}$$

The above process is 'back propagation'.

After doing this, diffrentiate with J respect to w of the perceptrons in the hiddenlayers can be convert to what is easy to calculate.