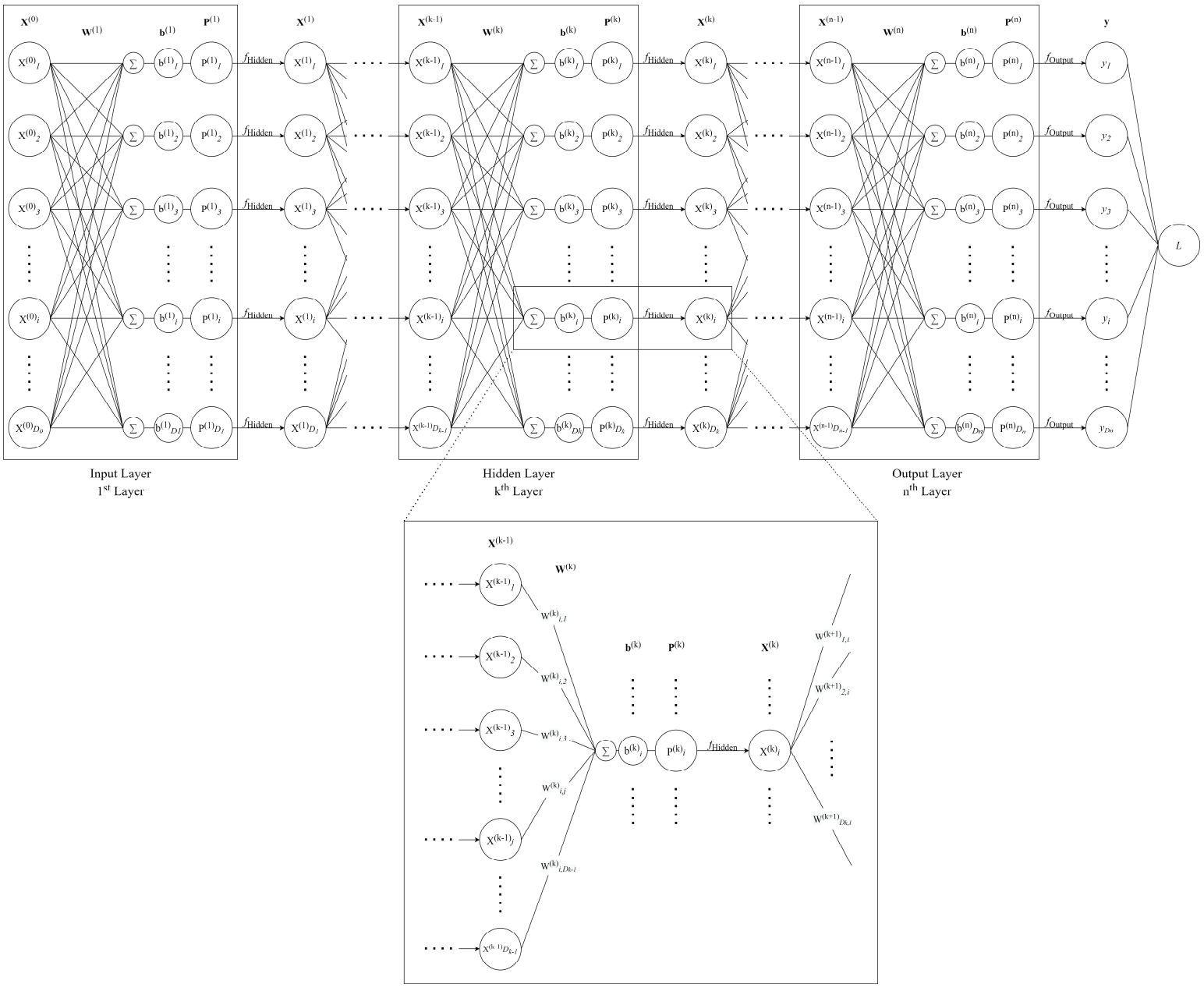
Assignment

Li Hantao, G2101725H, MSAI, hli038@e.ntu.edu.sg

We design a Full Connected Network with a fundamental representation in this assignment. Firstly, we will use this neural network structure to derive the gradients of the network parameters. In order to obtain a more general expression, in this step, we will use the symbol of function *f* instead of pointing out the specific function between hidden layers and the output objective function.



This is an n layer's fully connected neural network. Single-layer contains an input column vector $\mathbf{X}$, a weight matrix $\mathbf{W}$, a bias column vector $\mathbf{b}$, and a hidden layer column vector $\mathbf{P}$, whose element is used as an independent variable passing through the function f_{Hidden} to obtain the $\mathbf{X}$ of the next layer. In vector form, we can represent each component in a more comprehensible way:

$$\mathbf{X}^k = \begin{bmatrix} \mathrm{X}_1^k \\ \mathrm{X}_2^k \\ \vdots \\ \mathrm{X}_{D_k}^k \end{bmatrix} \quad \mathbf{W}^k = \begin{bmatrix} \mathrm{W}_{1,1}^k & \mathrm{W}_{1,2}^k & \cdots & \mathrm{W}_{1,D_{k-1}}^k \\ \mathrm{W}_{2,1}^k & \mathrm{W}_{2,2}^k & & \mathrm{W}_{2,D_{k-1}}^k \\ \vdots & & \ddots & \vdots \\ \mathrm{W}_{D_k,1}^k & \mathrm{W}_{D_k,2}^k & \cdots & \mathrm{W}_{D_k,D_{k-1}}^k \end{bmatrix} \quad \mathbf{b}^k = \begin{bmatrix} \mathrm{b}_1^k \\ \mathrm{b}_2^k \\ \vdots \\ \mathrm{b}_{D_k}^k \end{bmatrix} \quad \mathbf{P}^k = \begin{bmatrix} \mathrm{P}_1^k \\ \mathrm{P}_2^k \\ \vdots \\ \mathrm{P}_{D_k}^k \end{bmatrix}$$

The upper right corner k represents the position of the vector in the network. Here, we regard a group of networks composed of $\mathbf{X}^{k-1}$, $\mathbf{W}^k$, $\mathbf{b}^k$ and $\mathbf{P}^k$ as *k-th* networks, as shown in the figure. $\mathbf{X}^0$ is the input feature vector. The lower right corner represents the variables in the vector. We use n-dimensional vector $\mathbf{D} = [\mathrm{D}_0, \mathrm{D}_1, \cdots, \mathrm{D}_n]$ to store the number of neurons in each layer of neural network; that is, D_k is the number of neurons in *k-th* layer (D_0 is the dimension of the $\mathbf{X}^0$). The number of neurons in each layer in the figure is consistent only for the convenience of representation. In addition, the *i-th* variable in the vector in the figure is only convenient for representation as well; that is, the i of each layer is neither consistent nor of particular significance.

For the internal calculation of each layer of network, we have the following relationship (refer to the enlarged part in the figure):

$$\mathbf{P}^k = \mathbf{W}^k\mathbf{X}^{k-1} + \mathbf{b}^k = \begin{bmatrix} \mathrm{W}_{1,1}^k & \mathrm{W}_{1,2}^k & \cdots & \mathrm{W}_{1,D_{k-1}}^k \\ \mathrm{W}_{2,1}^k & \mathrm{W}_{2,2}^k & & \mathrm{W}_{2,D_{k-1}}^k \\ \vdots & & \ddots & \vdots \\ \mathrm{W}_{D_k,1}^k & \mathrm{W}_{D_k,2}^k & \cdots & \mathrm{W}_{D_k,D_{k-1}}^k \end{bmatrix} \begin{bmatrix} \mathrm{X}_1^{k-1} \\ \mathrm{X}_2^{k-1} \\ \vdots \\ \mathrm{X}_{D_{k-1}}^{k-1} \end{bmatrix} + \begin{bmatrix} \mathrm{b}_1^k \\ \mathrm{b}_2^k \\ \vdots \\ \mathrm{b}_{D_k}^k \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{W}_{1,1}^k \cdot \mathrm{X}_1^{k-1} + \mathrm{W}_{1,2}^k \cdot \mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{1,D_{k-1}}^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_1^k \\ \mathrm{W}_{2,1}^k \cdot \mathrm{X}_1^{k-1} + \mathrm{W}_{2,2}^k \cdot \mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{2,D_{k-1}}^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_2^k \\ \vdots \\ \mathrm{W}_{i,1}^k \cdot \mathrm{X}_1^{k-1} + \mathrm{W}_{i,2}^k \cdot \mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{i,D_{k-1}}^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_i^k \\ \vdots \\ \mathrm{W}_{D_k,1}^k \cdot \mathrm{X}_1^{k-1} + \mathrm{W}_{D_k,2}^k \cdot \mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{D_k,D_{k-1}}^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_{D_k}^k \end{bmatrix} = \begin{bmatrix} \sum_{\mathrm{j}=1}^{D_{k-1}} \mathrm{W}_{1,j}^k \cdot \mathrm{X}_j^{k-1} + \mathrm{b}_1^k \\ \sum_{\mathrm{j}=1}^{D_{k-1}} \mathrm{W}_{2,j}^k \cdot \mathrm{X}_j^{k-1} + \mathrm{b}_2^k \\ \vdots \\ \sum_{\mathrm{j}=1}^{D_{k-1}} \mathrm{W}_{i,j}^k \cdot \mathrm{X}_j^{k-1} + \mathrm{b}_i^k \\ \vdots \\ \sum_{\mathrm{j}=1}^{D_{k-1}} \mathrm{W}_{D_k,j}^k \cdot \mathrm{X}_j^{k-1} + \mathrm{b}_{D_k}^k \end{bmatrix} = \begin{bmatrix} \mathrm{P}_1^k \\ \mathrm{P}_2^k \\ \vdots \\ \mathrm{P}_i^k \\ \vdots \\ \mathrm{P}_{D_k}^k \end{bmatrix}$$

For the transfer calculation of the adjacent layer of the network, we utilize the transfer function f_{Hidden} to obtain the input for the next layer. Typically, f_{Hidden} is an element-wise function, such as **ReLU** and **Sigmoid**. In this step, we will not point out the particular function to simplify the subsequent derivation. Moreover, we will use f instead of f_{Hidden} to simplify formulas. We assume f is an element-wise function in our assignment; thus, we have the following relationship:

$$\mathbf{X}^{k+1} = f(\mathbf{P}^k) = \begin{bmatrix} f(\mathbf{P}^k)_1 \\ f(\mathbf{P}^k)_2 \\ \vdots \\ f(\mathbf{P}^k)_i \\ \vdots \\ f(\mathbf{P}^k)_{D_k} \end{bmatrix} = \begin{bmatrix} f(\mathrm{P}_1^k) \\ f(\mathrm{P}_2^k) \\ \vdots \\ f(\mathrm{P}_i^k) \\ \vdots \\ f(\mathrm{P}_{D_k}^k) \end{bmatrix}$$

$$\frac{\partial \mathbf{X}^{k+1}}{\partial \mathbf{P}^k} = \frac{\partial f(\mathbf{P}^k)}{\partial \mathbf{P}^k} = \begin{bmatrix} \frac{\partial f(\mathrm{P}_1^k)}{\partial \mathrm{P}_1^k} & \frac{\partial f(\mathrm{P}_1^k)}{\partial \mathrm{P}_2^k} & \cdots & \frac{\partial f(\mathrm{P}_1^k)}{\partial \mathrm{P}_{D_k}^k} \\ \frac{\partial f(\mathrm{P}_2^k)}{\partial \mathrm{P}_1^k} & \frac{\partial f(\mathrm{P}_2^k)}{\partial \mathrm{P}_2^k} & & \frac{\partial f(\mathrm{P}_2^k)}{\partial \mathrm{P}_{D_k}^k} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f(\mathrm{P}_{D_k}^k)}{\partial \mathrm{P}_1^k} & \frac{\partial f(\mathrm{P}_{D_k}^k)}{\partial \mathrm{P}_2^k} & \cdots & \frac{\partial f(\mathrm{P}_{D_k}^k)}{\partial \mathrm{P}_{D_k}^k} \end{bmatrix} = \begin{bmatrix} f'(\mathrm{P}_1^k) & 0 & \cdots & 0 \\ 0 & f'(\mathrm{P}_2^k) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & f'(\mathrm{P}_{D_k}^k) \end{bmatrix}$$

When using the *Gradient Descent* method to update the parameters, we need to update the weight matrix $\mathbf{W}$ and the bias coefficient vector $\mathbf{b}$ in each layer of the neural network; thus, $\partial\mathcal{L}/\partial\mathbf{W}$ and $\partial\mathcal{L}/\partial\mathbf{b}$ are required, where $\mathcal{L}$ is the error calculated through the loss function we designed, such as $\boldsymbol{l_2}$**-norm** loss function. The output of the last layer $\mathbf{P}^n$ will pass through an output function f_{Output}, such as **Softmax**, to generate the predicted label $\boldsymbol{y}$ of input. Then, the loss function will be used to quantify the error between the real label $\mathbf{Y}$ and the predicted label $\boldsymbol{y}$. We can express the above process with the following expression:

$$\boldsymbol{y} = f_{\text{Output}}(\mathbf{P}^n) \qquad \mathcal{L} = Loss(\mathbf{Y}, \boldsymbol{y})$$

With the chain rule, we can begin to calculate the gradient:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}^k} = \frac{\partial\mathcal{L}}{\partial\mathbf{P}^k}\cdot\frac{\partial\mathbf{P}^k}{\partial\mathbf{W}^k} \qquad \frac{\partial\mathcal{L}}{\partial\mathbf{b}^k} = \frac{\partial\mathcal{L}}{\partial\mathbf{P}^k}\cdot\frac{\partial\mathbf{P}^k}{\partial\mathbf{b}^k}$$

As shown above, both two gradients need the same component $\partial\mathcal{L}/\partial\mathbf{P}$; thus, we need to compute this first in order to obtain the gradient. We assume that $\partial\mathcal{L}/\partial\mathbf{P}^k = \boldsymbol{\delta}^k = [\delta_1^k, \delta_2^k, \cdots, \delta_{D_k}^k]$. We first calculate the single component δ_i^k of a hidden layer k (the output layer is easier to compute, which will be mentioned in the following section) for the convenience of calculation.

$$\begin{aligned}
\delta_i^k = \frac{\partial\mathcal{L}}{\partial \mathrm{P}_i^k} &= \frac{\partial\mathcal{L}}{\partial \mathrm{X}_i^k}\cdot\frac{\partial \mathrm{X}_i^k}{\partial \mathrm{P}_i^k} \\
&= \left(\frac{\partial\mathcal{L}}{\partial \mathrm{P}_1^{k+1}}\cdot\frac{\partial \mathrm{P}_1^{k+1}}{\partial \mathrm{X}_i^k} + \frac{\partial\mathcal{L}}{\partial \mathrm{P}_2^{k+1}}\cdot\frac{\partial \mathrm{P}_2^{k+1}}{\partial \mathrm{X}_i^k} + \cdots + \frac{\partial\mathcal{L}}{\partial \mathrm{P}_{D_{k+1}}^{k+1}}\cdot\frac{\partial \mathrm{P}_{D_{k+1}}^{k+1}}{\partial \mathrm{X}_i^k}\right)(f'(\mathrm{P}_i^k)) \\
&= \left(\sum_{j=1}^{D_{k+1}}\frac{\partial\mathcal{L}}{\partial \mathrm{P}_j^{k+1}}\cdot\frac{\partial \mathrm{P}_j^{k+1}}{\partial \mathrm{X}_i^k}\right)(f'(\mathrm{P}_i^k)) \\
&= \left(\sum_{j=1}^{D_{k+1}}\frac{\partial\mathcal{L}}{\partial \mathrm{P}_j^{k+1}}\cdot\frac{\partial\left(\sum_{\mathrm{m}=1}^{D_k}\mathrm{W}_{j,m}^{k+1}\cdot\mathrm{X}_m^k + \mathrm{b}_j^{k+1}\right)}{\partial \mathrm{X}_i^k}\right)(f'(\mathrm{P}_i^k)) \\
&= \left(\sum_{j=1}^{D_{k+1}}\frac{\partial\mathcal{L}}{\partial \mathrm{P}_j^{k+1}}\cdot\frac{\partial\left(\mathrm{W}_{j,1}^{k+1}\cdot\mathrm{X}_1^k + \mathrm{W}_{j,2}^{k+1}\cdot\mathrm{X}_2^k + \cdots + \mathrm{W}_{j,i}^{k+1}\cdot\mathrm{X}_i^k + \cdots + \mathrm{W}_{j,D_k}^{k+1}\cdot\mathrm{X}_{D_k}^k + \mathrm{b}_j^{k+1}\right)}{\partial \mathrm{X}_i^k}\right)(f'(\mathrm{P}_i^k)) \\
&= \left(\sum_{j=1}^{D_{k+1}}\delta_j^{k+1}\cdot\mathrm{W}_{j,i}^{k+1}\right)(f'(\mathrm{P}_i^k)) \\
&= (\boldsymbol{\delta}^{k+1}\cdot\mathbf{W}_{:,i}^{k+1})(f'(\mathrm{P}_i^k))
\end{aligned}$$

where $\mathbf{W}_{:,i}^{k+1}$ is the *i-th* column of the $\mathbf{W}^{k+1}$. We can observe that the single component δ_i^k is the combination of $\boldsymbol{\delta}^{k+1}$ and a single column of $\mathbf{W}^{k+1}$, which means that $\boldsymbol{\delta}^k$ corresponds to the matrix multiplication of $\boldsymbol{\delta}^{k+1}$ and $\mathbf{W}^{k+1}$, that is:

$$\boldsymbol{\delta}^k = (\boldsymbol{\delta}^{k+1}\cdot\mathbf{W}^{k+1}) \odot (f'(\mathbf{P}^k))$$

where $\odot$ is the element-wise symbol. Then, we can obtain the single component of gradients with the δ_i^k:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial \mathrm{W}_{i,j}^k} = \frac{\partial\mathcal{L}}{\partial \mathrm{P}_i^k}\cdot\frac{\partial \mathrm{P}_i^k}{\partial \mathrm{W}_{i,j}^k} &= \delta_i^k\cdot\frac{\mathrm{W}_{i,1}^k\cdot\mathrm{X}_1^{k-1} + \mathrm{W}_{i,2}^k\cdot\mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{i,j}^k\cdot\mathrm{X}_i^{k-1} + \cdots + \mathrm{W}_{i,D_{k-1}}^k\cdot\mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_i^k}{\partial \mathrm{W}_{i,j}^k} = \delta_i^k\cdot\mathrm{X}_i^{k-1} \\
\frac{\partial\mathcal{L}}{\partial \mathrm{b}_i^k} = \frac{\partial\mathcal{L}}{\partial \mathrm{P}_i^k}\cdot\frac{\partial \mathrm{P}_i^k}{\partial \mathrm{b}_i^k} &= \delta_i^k\cdot\frac{\mathrm{W}_{i,1}^k\cdot\mathrm{X}_1^{k-1} + \mathrm{W}_{i,2}^k\cdot\mathrm{X}_2^{k-1} + \cdots + \mathrm{W}_{i,j}^k\cdot\mathrm{X}_i^{k-1} + \cdots + \mathrm{W}_{i,D_{k-1}}^k\cdot\mathrm{X}_{D_{k-1}}^{k-1} + \mathrm{b}_i^k}{\partial \mathrm{b}_i^k} = \delta_i^k
\end{aligned}$$

Now, we can complete the representation in the matrix expression. In order to maintain the shape of the weight matrix for the convenience of computing the gradient descent, we will use the **Denominator Layout Jacobian** in the weight matrix gradient, with the transposing to fit it. Moreover, the initial states ($\boldsymbol{\delta}^n$) and final states ($\partial\mathcal{L}/\partial\mathbf{X}^0$) of the gradient calculation are also given.

$$\boldsymbol{\delta}^n = \frac{\partial \mathcal{L}}{\partial \mathbf{P}^n} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \mathbf{P}^n} = \frac{\partial Loss(\mathbf{Y}, \boldsymbol{y})}{\partial \boldsymbol{y}} \odot (f_{\text{Output}}{}'(\mathbf{P}^n))$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^k} = \frac{\partial \mathcal{L}}{\partial \mathbf{P}^k} \cdot \frac{\partial \mathbf{P}^k}{\partial \mathbf{W}^k} = (\boldsymbol{\delta}^k)^T \cdot (\mathbf{X}^{k-1})^T$$

$$= \begin{bmatrix} \delta_1^k \\ \delta_2^k \\ \vdots \\ \delta_{D_k}^k \end{bmatrix} [\mathrm{X}_1^{k-1}, \mathrm{X}_2^{k-1}, \cdots, \mathrm{X}_{D_{k-1}}^{k-1}] = \begin{bmatrix} \delta_1^k \cdot \mathrm{X}_1^{k-1} & \delta_1^k \cdot \mathrm{X}_2^{k-1} & \cdots & \delta_1^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} \\ \delta_2^k \cdot \mathrm{X}_1^{k-1} & \delta_2^k \cdot \mathrm{X}_2^{k-1} & \cdots & \delta_2^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{D_k}^k \cdot \mathrm{X}_1^{k-1} & \delta_{D_k}^k \cdot \mathrm{X}_2^{k-1} & \cdots & \delta_{D_k}^k \cdot \mathrm{X}_{D_{k-1}}^{k-1} \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^k} = \frac{\partial \mathcal{L}}{\partial \mathbf{P}^k} \cdot \frac{\partial \mathbf{P}^k}{\partial \mathbf{b}^k} = (\boldsymbol{\delta}^k)^T$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}^0} = \frac{\partial \mathcal{L}}{\partial \mathbf{P}^1} \cdot \frac{\partial \mathbf{P}^1}{\partial \mathbf{X}^0} = (\boldsymbol{\delta}^1 \cdot \mathbf{W}^1)^T$$

With the Gradient descent updating method, we can also obtain the expression of training equations. α is the learning rate; (s) and (s+1) is the state indicator; that is, (s) is the current state, (s+1) is the updating state.

$$\mathbf{W}_{(s+1)}^k = \mathbf{W}_{(s)}^k - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(s)}^k} = \mathbf{W}_{(s)}^k - \alpha \cdot (\boldsymbol{\delta}^k)^T \cdot (\mathbf{X}^{k-1})^T$$

$$\mathbf{b}_{(s+1)}^k = \mathbf{b}_{(s)}^k - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{b}_{(s)}^k} = \mathbf{b}_{(s)}^k - \alpha \cdot (\boldsymbol{\delta}^k)^T$$

The final result of gradients computation and training equations based on gradient descent are shown below:

$$\begin{cases} \boldsymbol{\delta}_{(s)}^n = \dfrac{\partial Loss(\mathbf{Y}, \boldsymbol{y}_{(s)})}{\partial \boldsymbol{y}_{(s)}} \odot (f_{\text{Output}}{}'(\mathbf{P}_{(s)}^n)) \\ \boldsymbol{\delta}_{(s)}^k = (\boldsymbol{\delta}_{(s)}^{k+1} \cdot \mathbf{W}_{(s)}^{k+1}) \odot (f'(\mathbf{P}_{(s)}^k)) \\ \dfrac{\partial \mathcal{L}}{\partial \mathbf{W}_{(s)}^k} = (\boldsymbol{\delta}_{(s)}^k)^T \cdot (\mathbf{X}_{(s)}^{k-1})^T \\ \dfrac{\partial \mathcal{L}}{\partial \mathbf{b}_{(s)}^k} = (\boldsymbol{\delta}_{(s)}^k)^T \\ \dfrac{\partial \mathcal{L}}{\partial \mathbf{X}^0} = (\boldsymbol{\delta}_{(s)}^1 \cdot \mathbf{W}_{(s)}^1)^T \\ \mathbf{W}_{(s+1)}^k = \mathbf{W}_{(s)}^k - \alpha \cdot \dfrac{\partial \mathcal{L}}{\partial \mathbf{W}_{(s)}^k} \\ \mathbf{b}_{(s+1)}^k = \mathbf{b}_{(s)}^k - \alpha \cdot \dfrac{\partial \mathcal{L}}{\partial \mathbf{b}_{(s)}^k} \end{cases}$$

Now, we specify the function in the network to show a more particular result. We specify the transfer function between the hidden layer f_{Hidden} as **Sigmoid**, the output function f_{Output} as **Softmax**, and the cost function $Loss(\mathbf{Y}, \boldsymbol{y})$ as the **Cross-Entropy Error Function** (for the convenience of calculation, we use the logarithm based on *e*)**:**

$$\begin{cases} f_{\text{Hidden}} := Sigmoid(\mathrm{P}_i^k) \\ f_{\text{Output}} := Softmax(\mathbf{P}^n) \\ Loss(\mathbf{Y}, \boldsymbol{y}) := CrossEntropy(\mathbf{Y}, \boldsymbol{y}) \end{cases} \rightarrow \begin{cases} \mathrm{X}_i^k = \dfrac{1}{1 + e^{-\mathrm{P}_i^k}} \\ y_i = \dfrac{e^{\mathrm{P}_i^n}}{\sum_{j=1}^{D_n} e^{\mathrm{P}_j^n}} \\ \mathcal{L} = \sum_{i=1}^{D_n} Y_i \log y_i \end{cases}$$

With those functions, we can continue calculating the particular expressions in our neural network.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial y_1} & \frac{\partial \mathcal{L}}{\partial y_2} & \cdots & \frac{\partial \mathcal{L}}{\partial y_{D_n}} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\partial (\sum_{i=1}^{D_n} Y_i \log y_i)}{\partial y_1} & \frac{\partial (\sum_{i=1}^{D_n} Y_i \log y_i)}{\partial y_2} & \cdots & \frac{\partial (\sum_{i=1}^{D_n} Y_i \log y_i)}{\partial y_{D_n}} \end{bmatrix}$$
$$= \begin{bmatrix} -\frac{Y_1}{y_1} & -\frac{Y_2}{y_2} & \cdots & -\frac{Y_{D_n}}{y_{D_n}} \end{bmatrix}$$

$$\frac{\partial \boldsymbol{y}}{\partial \mathbf{P}^n} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathrm{P}_1^n} & \frac{\partial y_1}{\partial \mathrm{P}_2^n} & \cdots & \frac{\partial y_1}{\partial \mathrm{P}_{D_k}^n} \\ \frac{\partial y_2}{\partial \mathrm{P}_1^n} & \frac{\partial y_2}{\partial \mathrm{P}_2^n} & & \frac{\partial y_2}{\partial \mathrm{P}_{D_k}^n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial y_{D_n}}{\partial \mathrm{P}_1^n} & \frac{\partial y_{D_n}}{\partial \mathrm{P}_2^n} & \cdots & \frac{\partial y_{D_n}}{\partial \mathrm{P}_{D_k}^n} \end{bmatrix} = \begin{bmatrix} y_1(1-y_1) & -y_1 y_2 & \cdots & -y_1 y_{D_n} \\ -y_1 y_2 & y_2(1-y_2) & \cdots & -y_2 y_{D_n} \\ \vdots & \vdots & \ddots & \vdots \\ -y_1 y_{D_n} & -y_2 y_{D_n} & \cdots & y_{D_n}(1-y_{D_n}) \end{bmatrix}$$

$$\boldsymbol{\delta}^n = \frac{\partial \mathcal{L}}{\partial \mathbf{P}^n} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \mathbf{P}^n} = \begin{bmatrix} -y_1(1-y_1) + y_1 Y_2 + y_1 Y_3 + \cdots + y_1 Y_{D_n} \\ y_2 Y_1 - y_2(1-y_2) + y_2 Y_3 + \cdots + y_2 Y_{D_n} \\ \vdots \\ y_i Y_1 + y_i Y_2 + \cdots + y_i Y_{i-1} - y_i(1-y_i) + y_i Y_{i+1} + \cdots + y_i Y_{D_n} \\ \vdots \\ y_{D_n} Y_1 + y_{D_n} Y_2 + \cdots + y_{D_n - 1} Y_{D_n} - y_{D_n}(1-y_{D_n}) \end{bmatrix}^T$$
$$= \begin{bmatrix} -Y_1 + y_1 \cdot \sum_i Y_i & -Y_2 + y_2 \cdot \sum_i Y_i & \cdots & -Y_{D_n} + y_{D_n} \cdot \sum_i Y_i \end{bmatrix} \quad \left(\sum_i Y_i = 1\right)$$
$$= \begin{bmatrix} y_1 - Y_1 & y_2 - Y_2 & \cdots & y_{D_n} - Y_{D_n} \end{bmatrix}$$
$$= (\boldsymbol{y} - \mathbf{Y})^T$$

$$(f'(\mathrm{P}_i^k)) = \left(\frac{1}{1+e^{-\mathrm{P}_i^k}}\right)' = \frac{-e^{-\mathrm{P}_i^k}}{(1+e^{-\mathrm{P}_i^k})^2} = \frac{1}{1+e^{-\mathrm{P}_i^k}} \cdot \frac{e^{-\mathrm{P}_i^k}}{1+e^{-\mathrm{P}_i^k}} = f(\mathrm{P}_i^k) \cdot (1 - f(\mathrm{P}_i^k))$$

$$\delta_i^k = (\boldsymbol{\delta}^{k+1} \cdot \mathbf{W}_{:,i}^{k+1})(f'(\mathrm{P}_i^k)) = (\boldsymbol{\delta}^{k+1} \cdot \mathbf{W}_{:,i}^{k+1})\left(\frac{1}{1+e^{-\mathrm{P}_i^k}} \cdot \frac{e^{-\mathrm{P}_i^k}}{1+e^{-\mathrm{P}_i^k}}\right)$$

So far, we have derived all the information we need. For the state stored in each epoch, we can use the calculated loss function to deduce $\boldsymbol{\delta}_{(s)}^n$, and further obtain the gradient $\partial \mathcal{L}/\partial \mathbf{W}_{(s)}^n$ and $\partial \mathcal{L}/\partial \mathbf{b}_{(s)}^n$. Then, the $\boldsymbol{\delta}_{(s)}^{n-1}$, $\partial \mathcal{L}/\partial \mathbf{W}_{(s)}^{n-1}$, $\partial \mathcal{L}/\partial \mathbf{b}_{(s)}^{n-1}$.....Until we gradually get the corresponding gradient of each layer of the network, at this time, the **Backpropagation** is completed. Then, we can use the gradient descent method to compute each $\mathbf{W}_{(s+1)}^k$ and $\mathbf{b}_{(s+1)}^k$ to obtain the new weight matrix and bias vector for the next epoch. In addition, although we use the same output function and loss function equation as in Tutorial 6 in this assignment, it is not because of laziness but to simplify $\boldsymbol{\delta}^n$. When we use other loss functions, such as the $\boldsymbol{l_2}$ **norm**, the component of the derivative of output function cannot be combined and simplified with the vector of the loss function, and the result will be extremely complex:

$$\frac{\partial Loss\ of\ \boldsymbol{l_2}\ \boldsymbol{norm}}{\partial \boldsymbol{y}} = \begin{bmatrix} -(Y_1 - y_1) & -(Y_2 - y_2) & \cdots & -(Y_{D_n} - y_{D_n}) \end{bmatrix}$$

The first component of $\boldsymbol{\delta}^n$ will become $({y_1}^2 - Y_1 y_1 + y_1(Y_1 y_1 + Y_2 y_2 + \cdots + Y_{D_n} y_{D_n} - {y_1}^2 - {y_2}^2 - \cdots - {y_{D_n}}^2))$, instead of $(y_1 - Y_1)$. Generally, the solution of simplifying the $\boldsymbol{l_2}$ **norm** loss function is to specify the f_{Output} as the element-wise function, such as **Sigmoid**. However, it is as same as the f_{Hidden} in terms of method and can not achieve the significance of distinction. Consequentially, we utilize the above function. Nevertheless, with the universal expression we obtained, you can substitute any suitable function equations, such as **ReLU**, **Tanh**, and **Softplus**.