# Report for Lab2-part 2

Name: Wu Yihang

StuID: A0285643W

Reference :In this part, the code is based on Lecture 6 Slides Page. 43.

## _learn_node_parameter_w()

In this function, we get $w_{u_0}, w_{u_1}, \ldots, w_{u_n}$. We can adjust the formula in the slides Page. 43 to become a solvable form of the function `np.linalg.solve()`. As `np.linalg.solve(A,b)` will calculate X for `AX = b`, in our case `X` is $[w_{u_0}, w_{u_1}, \ldots, w_{u_n}]^T$. So the problem becomes to get `A` and `b`. We can change the equation in slides into a matrix product-friendly form, take $\frac{\partial L}{\partial w_{u0}}$ as an example:

$$\sum_{n=1}^{N}(w_{u1}x_{u1,n}+\ldots+w_{uC}x_{uC,n}+w_{u0}\cdot 1)\cdot 1=\sum_{n=1}^{N}x_{u,n} \tag{2}$$

Then by listing all the $[w_{u_0}, w_{u_1}, \ldots, w_{u_n}]^T$, we can modify it into an `AX=b` matrix form:

$$\begin{bmatrix} \sum_{n=1}^{N}1\cdot 1 & \sum_{n=1}^{N}1\cdot x_{u1,n} & \sum_{n=1}^{N}1\cdot x_{u2,n} & \cdots & \sum_{n=1}^{N}1\cdot x_{uC,n} \\ \sum_{n=1}^{N}1\cdot x_{u1,n} & \sum_{n=1}^{N}x_{u1,n}x_{u1,n} & \sum_{n=1}^{N}x_{u1,n}x_{u2,n} & \cdots & \sum_{n=1}^{N}x_{u1,n}x_{uC,n} \\ \sum_{n=1}^{N}1\cdot x_{u2,n} & \sum_{n=1}^{N}x_{u2,n}x_{u1,n} & \sum_{n=1}^{N}x_{u2,n}x_{u2,n} & \cdots & \sum_{n=1}^{N}x_{u2,n}x_{uC,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{n=1}^{N}1\cdot x_{uC,n} & \sum_{n=1}^{N}x_{uC,n}x_{u1,n} & \sum_{n=1}^{N}x_{uC,n}x_{u2,n} & \cdots & \sum_{n=1}^{N}x_{uC,n}x_{uC,n} \end{bmatrix} \begin{bmatrix} w_{u_0} \\ w_{u_1} \\ \vdots \\ w_{u_C} \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{N}x_{u,n} \\ \sum_{n=1}^{N}x_{u,n} \\ \vdots \\ \sum_{n=1}^{N}x_{u,n} \end{bmatrix} \tag{3}$$

As there is 1 in equations, we can expand our input, adding 1 to its first column, expanding its dimension to $N\times(I+1)$, then we can build A with dimension $(I+1, I+1, N)$, where the third dimension is used for sum operation. Then we can get A by 3-loop iteration and a sum-up operation:

```
# A (I+1,I+1,N) x_ucn (N, I+1)
for i,weight in enumerate(A):
    for j,item in enumerate(weight):
        for k,to_sum_up in enumerate(item):
            A[i][j][k] = x_ucn[k][j]*x_ucn[k][i]
 A = np.sum(A,axis=2)
```

The matrix b is easy to get, then weights can get via `np.linalg.solve(A,b)`

## _learn_node_parameter_var()

For this part, we need to calculate the derivative of $L$ w.r.t to $\sigma^2$, and let it equal to zero

$$\frac{\partial L}{\partial \sigma^2}\sum_{n=1}^{N}\{-\frac{1}{2}\log(2\pi\sigma_u^2)-\frac{1}{2\sigma_u^2}\left(x_{u,n}-\left(\sum_{c\in x_{\pi_u}}w_{uc}x_{uc,n}+w_{u0}\right)\right)\}=-\frac{N}{2\sigma_u^2}+\frac{1}{(2\sigma_u^2)^2}\sum_{n=1}^{N}\left(x_{u,n}-\left(\sum_{c\in x_{\pi_u}}w_{uc}x_{uc,n}+w_{u0}\right)\right)=0$$

Then move $\sigma^2$ to one side, modify the equation to

$$\sigma^2=\sum_{n=1}^{N}\left(x_{u,n}-\left(\sum_{c\in x_{\pi_u}}w_{uc}x_{uc,n}+w_{u0}\right)\right)/N \tag{5}$$

Then we can write code based on this formula:

```
x_un = outputs
sum_c = None
if inputs is None:
    sum_c = 0
else:
    sum_c = np.sum(weights[1:]*inputs,axis=1)
var =  np.sum((x_un-(sum_c+weights[0]))**2)/len(x_un)
```

## _get_learned_parameters()

In this function, `nx` is used to build the graph and get the parent nodes of the current node, and then we can get the output and input of the current node and call the former 2 functions in loop iteration.