1 Introduction

This is an introduction of the theory of MLP, which will be illustrated by derivation with both MSE (Sigmoid) and cross entropy (softmax).

2 Derivation Process

2.1 Forward Propagation

Forward propagation can be understood as the process of network prediction, that is, forward propagation can transform the input information into the classification results. The schematic diagram is shown in the Fig. 1.

Suppose the input layer information is $[x_1, x_2, x_3]$, for layer l, L_i is used to represent all the neurons of the layer, and the output is y_l, where the output of the jth node is $y_l^{(j)}$, the input of the node is $u_l^{(j)}$, the weight matrix connecting layer l and layer $l-1$ is W_l, and the weight from the ith node of the previous layer $(l-1)$ to the jth node of layer l is $w_l^{(ji)}$. Therefore the output y_i can be written as:

$$\begin{cases} y_2^{(1)} = f(u_2^{(1)}) = f(\sum_{i=1}^{n} w_2^{1i} x_i + b_2^{(1)}) \\ y_2^{(2)} = f(u_2^{(2)}) = f(\sum_{i=1}^{n} w_2^{2i} x_i + b_2^{(2)}) \end{cases} \tag{1}$$

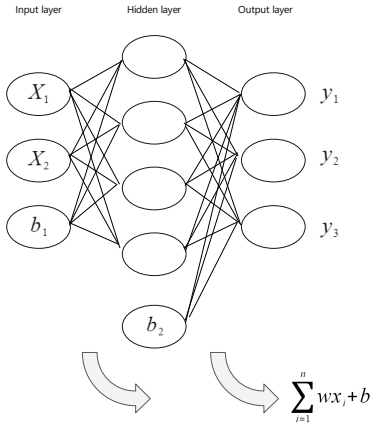

Figure 1: The schematic diagram of MLP

To make it easy to code, the first equation can be written as below:

$$y_2 = \begin{bmatrix} y_2^{(1)} \\ y_2^{(2)} \end{bmatrix} = f\left(\begin{bmatrix} w_2^{11} & w_2^{12} & w_2^{13} \\ w_2^{21} & w_2^{22} & w_2^{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_2^{(1)} \\ b_2^{(2)} \end{bmatrix} \right) = f(W_2 X + b_2) \tag{2}$$

Extend the forward propagation calculation process of the second layer to any layer in the network, then:

$$\begin{cases} y_l^{(j)} &= f(u_l^{(j)}) \\ u_l^{(j)} &= \sum_{i \in L_{l-1}} w_l^{ji} y_{l-1}^{(i)} + b_l^j \\ y_l &= f(u_l) \;=\; f(W_l y_{l-1} + b_l) \end{cases} \tag{3}$$

where $f(\cdot)$ is activation function and $b_l^{(j)}$ is the bias of the jth node in layer l.

2.2 Backward Propagation

After the basic model is built, the process of training is to update the model parameters. Due to the multi-layer network structure, it is not possible to directly update the parameters of the hidden layer by the loss function, but the back propagation of the loss from the top layer to the bottom layer can be used to estimate the parameters. Suppose that there are multiple neurons in the output layer of the multi-layer perceptron, and each neuron corresponds to a label. Input sample is $x = [x_1, x_2, \vdots x_n]$ and the label is t.

For the definition of the loss function of the output layer, this article includes:

$$E_{MSE} = \frac{1}{2}\sum_{j \in L_k}(t^j - y_k^{(j)})^2 \tag{4}$$

or

$$E_{Cross\ Entropy} = -\sum t^j \cdot log(y_k^{(j)}) \tag{5}$$

The loss function determines the problem of updating the parameters of the output layer. In order to better improve the training accuracy, the loss function with sigmoid as the activation function of the output layer is defined as MSE, while the loss function with softmax as the activation function of the output layer is defined as the cross entropy loss function.

In order to minimize the loss function, it is derived by gradient descent.

$$\begin{cases} \frac{\partial E}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial u_l^{(j)}}\frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} \\ \frac{\partial E}{\partial b_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial b_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial u_l^{(j)}}\frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} \end{cases} \tag{6}$$

It is easy to derive the following formulas:

$$\begin{cases} \frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} = f'(u_l^j) \\ \frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = y_{l-1}^{(i)} \\ \frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = 1 \end{cases} \tag{7}$$

Therefore,

$$\begin{cases} \frac{\partial E}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial u_l^{(j)}}\frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial y_l^{(j)}}f'(u_l^j)y_{l-1}^{(i)} \\ \frac{\partial E}{\partial b_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}}\frac{\partial y_l^{(j)}}{\partial u_l^{(j)}}\frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}}f'(u_l^j) \end{cases} \tag{8}$$

Also, each nodes of the next layer is related to all of the nodes from the former layer, and the loss function can be considered as the function of the input of each nodes from next layer, which means:

$$\begin{aligned} \frac{\partial E}{\partial w_l^{(ji)}} &= \frac{\partial E(u_{l+1}^{(1)}, u_{l+1}^{(2)}, \ldots, u_{l+1}^{(k)}, \ldots, u_{l+1}^{(K)})}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial u_{l+1}^k}\frac{\partial u_{l+1}^k}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^k}\frac{\partial y_{l+1}^k}{\partial u_{l+1}^k}\frac{\partial u_{l+1}^k}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^k}\frac{\partial y_{l+1}^k}{\partial u_{l+1}^k}w_{l+1}^{kj} \end{aligned} \tag{9}$$

For better understanding of calculation, define $\delta = \frac{\partial E}{\partial u}$ as the rate of change of error to input, which is node sensitivity. Therefore, the node sensitivity of the jth node from layer l is:

$$\delta_l^{(j)} = \frac{\partial E}{\partial u_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}} \frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) \tag{10}$$

Apply the node sensitivity to the derivation of loss function, it can be simplified as:

$$\begin{aligned} \frac{\partial E}{\partial w_l^{(ji)}} &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^k} \frac{\partial y_{l+1}^k}{\partial u_{l+1}^k} w_{l+1}^{kj} \\ &= \sum_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{kj} \end{aligned} \tag{11}$$

However, in view of the output layer, there is no "next layer" to be considered, and the output layer does not fit the equation above. The output of the final layer is related to the error, and the derivation of the loss function can be considered directly. Therefore, multiply $f'(u_l^{(j)})$ to both sides and the equation can be written as:

$$\delta_l^{(j)} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) = \begin{cases} f'(u_l^{(j)}) \sum\limits_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{kj} & if\ l\ is\ hidden\ layer \\ f'(u_l^{(j)}) \frac{\partial E}{\partial y_l^{(j)}} & if\ l\ is\ output\ layer \end{cases} \tag{12}$$

The gradient of loss function to each parameter is:

$$\begin{cases} \frac{\partial E}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \delta_l^{(j)} y_{l-1}^{(i)} \\ \frac{\partial E}{\partial b_l^{(j)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \delta_l^{(j)} \end{cases} \tag{13}$$

To make it suitable for every nodes and easy to code, the matrix format of equation above can be as $\frac{\partial E}{\partial W_l} = \delta_l y_{l-1}^T$ and $\frac{\partial E}{\partial b_l} = \delta_l$, where:

$$\delta_l = \begin{cases} (W_{l+1}^T \delta_{l+1}) \circ f'(u_l), & if\ l\ is\ hidden\ layer \\ \frac{\partial E}{\partial y_l} \circ f'(u_l), & if\ l\ is\ output\ layer \end{cases} \tag{14}$$

The $\circ$ represents the multiplication of corresponding elements in a matrix or vector. And the update equation of weights from each layers are:

$$\begin{cases} W_l = W_l - \eta \frac{\partial E}{\partial W_l} = W_l - \eta \delta_l y_{l-1}^T \\ b_l = b_l - \eta \frac{\partial E}{\partial b} = b_l - \eta \delta_l \end{cases} \tag{15}$$

2.3 Derivation of Sigmoid and MSE

Sigmoid function can be simplified as:

$$y_l = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - (e^x+1)^{-1} \tag{16}$$

Therefore, the derivation of sigmoid function is:

$$(y_l)' = (-1)(-1)(e^x+1)^2 e^x \quad = (e^x+1)^{-2} e^{-2x} e^x = (e^x+1)^{-1} \frac{1}{1+e^{-x}} \quad = y_l(1-y_l) \tag{17}$$

Referring to the E_{MSE} equation above, the derivation of MSE loss function is:

$$\frac{\partial E}{\partial y_l^{(j)}} = y_l^{(j)} - t^{(j)} \tag{18}$$

2.3.1 Derivation of Softmax and Cross Entropy

The derivation of softmax and cross entropy can be considered as $\frac{\partial E}{\partial u_i} = \sum_j (\frac{\partial E_j}{\partial y_j} \frac{\partial y_j}{\partial u_i})$. For the first derivation $\frac{\partial E_j}{\partial y_j}$:

$$\frac{\partial E_j}{\partial y_j} = \frac{\partial(-t_j ln y_j)}{\partial y_j} = -t_j \frac{1}{y_j} \tag{19}$$

As for the softmax part, there are two conditions. When $i = j$:

$$\frac{\partial y_j}{\partial u_i} = \frac{\partial(\frac{e^{z_i}}{\sum_k e^{z_k}})}{\partial z_i} = \frac{\sum_k e^{z_k} e^{z_i} - (e^{z_i})^2}{(\sum_k e^{z_k})^2} = (\frac{e^{z_i}}{\sum_k e^{z_k}})(1 - \frac{e^{z_i}}{\sum_k e^{z_k}}) = y_j(1 - y_j) \tag{20}$$

However, if $i \neq j$:

$$\frac{\partial y_j}{\partial u_i} = \frac{\partial(\frac{e^{z_i}}{\sum_k e^{z_k}})}{\partial z_i} = -e^{z_j}(\frac{1}{\sum_k e^{z_k}})^2 e^{z_i} = -y_i y_j \tag{21}$$

Combine the equations above, the final derivation of softmax and cross entropy is:

$$\begin{aligned}
\frac{\partial E}{\partial u_i} &= \sum_j (\frac{\partial E_j}{\partial y_j} \frac{\partial y_j}{\partial u_i}) = \sum_{j \neq i} (\frac{\partial E_j}{\partial y_j} \frac{\partial y_j}{\partial u_i}) + \sum_{j=i} (\frac{\partial E_j}{\partial y_j} \frac{\partial y_j}{\partial u_i}) \\
&= \left(\sum_{j \neq i} -t_j \frac{1}{y_j} (-y_i y_j) \right) + (-t_i \frac{1}{y_i})(y_i(1 - y_i)) \\
&= \left(\sum_{j \neq i} y_i t_j \right) + (-t_i(1 - y_i)) \\
&= \left(\sum_{j \neq i} y_i t_j \right) + y_i t_i - t_i \\
&= \left(y_i \sum_{j=i} t_j \right) - t_i
\end{aligned} \tag{22}$$

There is only one t_i will be 1, therefore, the $\sum_{j=i} t_j = 1$. In the end, the final $\frac{\partial E}{\partial u_i} = y_i - t_i$.

2.3.2 Problems of Coding Sigmoid and Softmax

The program overflowed when designing sigmoid and softmax methods in this project. In order to solve the above problems, the properties of the two activation functions are studied in this project. The original form of sigmoid function is $\sigma(z) = \frac{1}{1-e^{-z}}$. However, when the input Z is negative and its absolute value is relatively small, it is easy to cause overflow. To solve this problem, the equation code of sigmoid is modified as follows in this project.

```
def sigmoid_function(input):
    fz = []
    input = input.tolist()
    for num in input:
        if num >= 0:
            fz.append(1.0 / (1 + math.exp(-num)))
        else:
            fz.append(math.exp(num) / (1 + math.exp(num)))
    output = np.array(fz)
    return output
```

Similarly, the original form of softmax is $y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$. However, if the value of the input signal is too large, the program will overflow. To avoid this, the softmax formula can be modified

as follows without changing the result of the activation function: $y_k = \frac{e^{a_k - C}}{\sum_{i=1}^{n} e^{a_i - C}}$, where C is the max number in the input array. The corresponding code is shown as below:

```
def softmax_function(input):
  max_input = np.max(input)
  input = np.exp(input-max_input)
  sum_input = np.sum(input)
  output = input/sum_input
  return output
```