**SoftMax for Mutli-Class Classification**

1. In the intersection point, the probabilities of the x belong to Class 1, Class 2 and Class 3 is equal:

***p(c1|x) = p(c2|x) = p(c3|x) = 1 / 3***

1. The probabilities of the points along the red line that belongs to the two classes are equal. For instance, the points in the red line divides the class 1 and class 2:

***p(c1|x) = p(c1|x)***

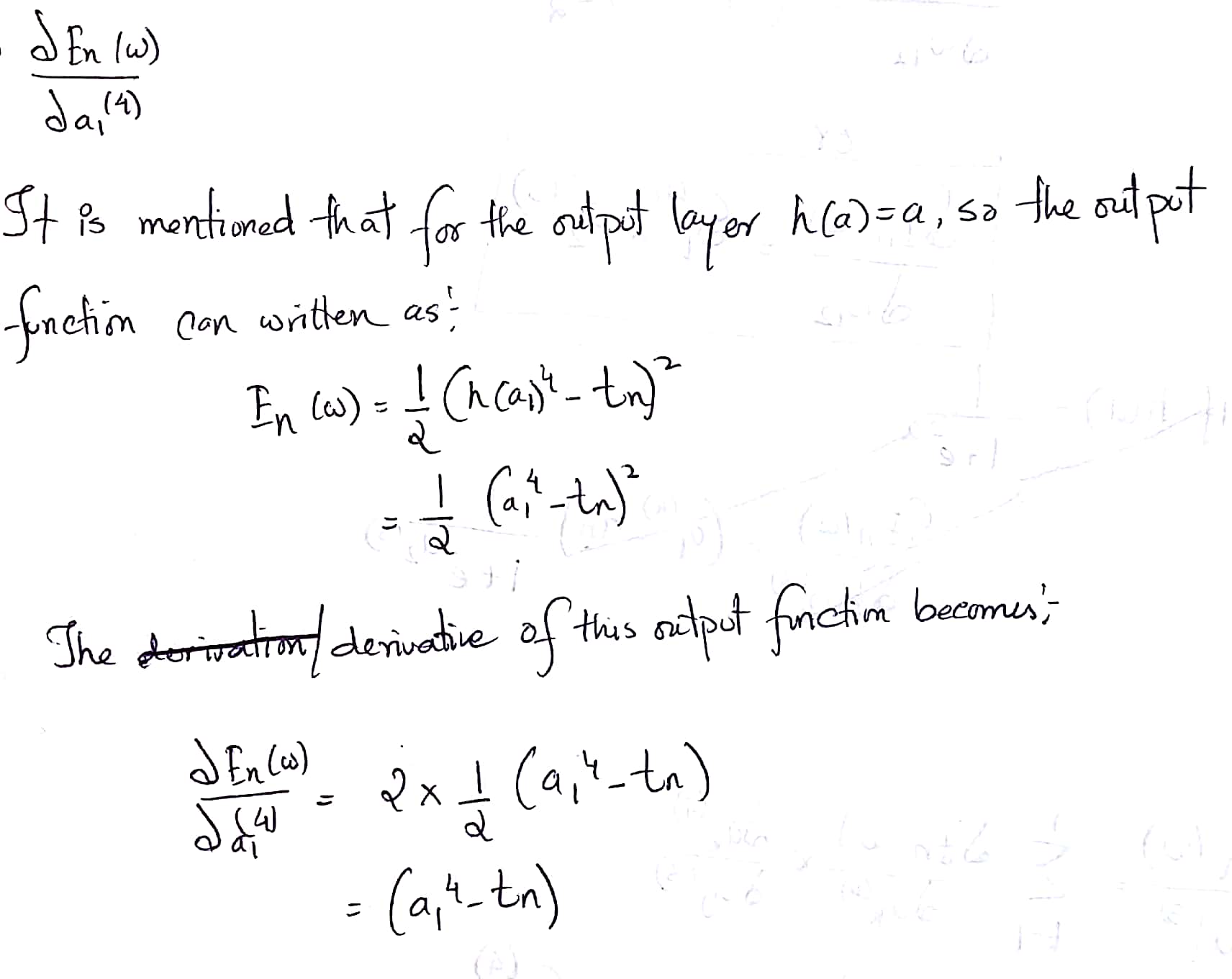
Subsequently, the probabilities of the points belonging to the third class decreases as the probability of the other two classes increases. In this example, ***p(c1|x), p(c2|x)*** will increase and ***p(c3|x)*** will decrease.

1. If we move far away from the intersection point and staying in the middle of one region, the probability for the points belonging to this region will increase and it will be larger than that of the remaining two regions i.e. decrease as the point moves. For example, if we stay in the region 1 (Class 1):

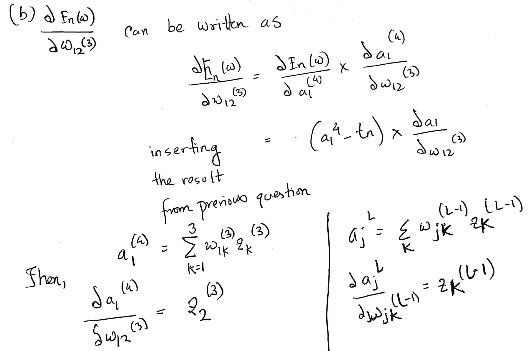
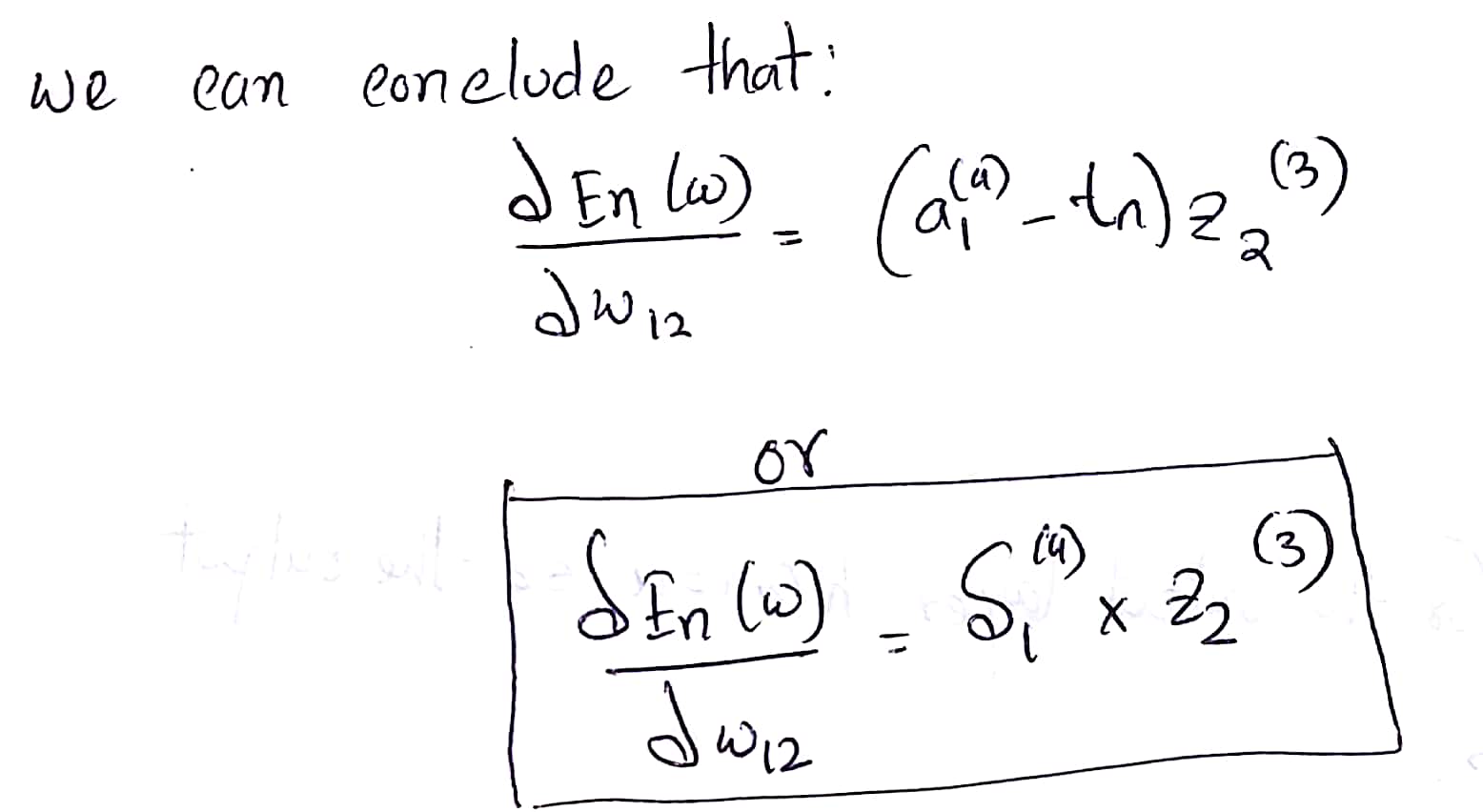
***p(c1|x) > p(c2|x)***

***p(c1|x) > p(c3|x)***

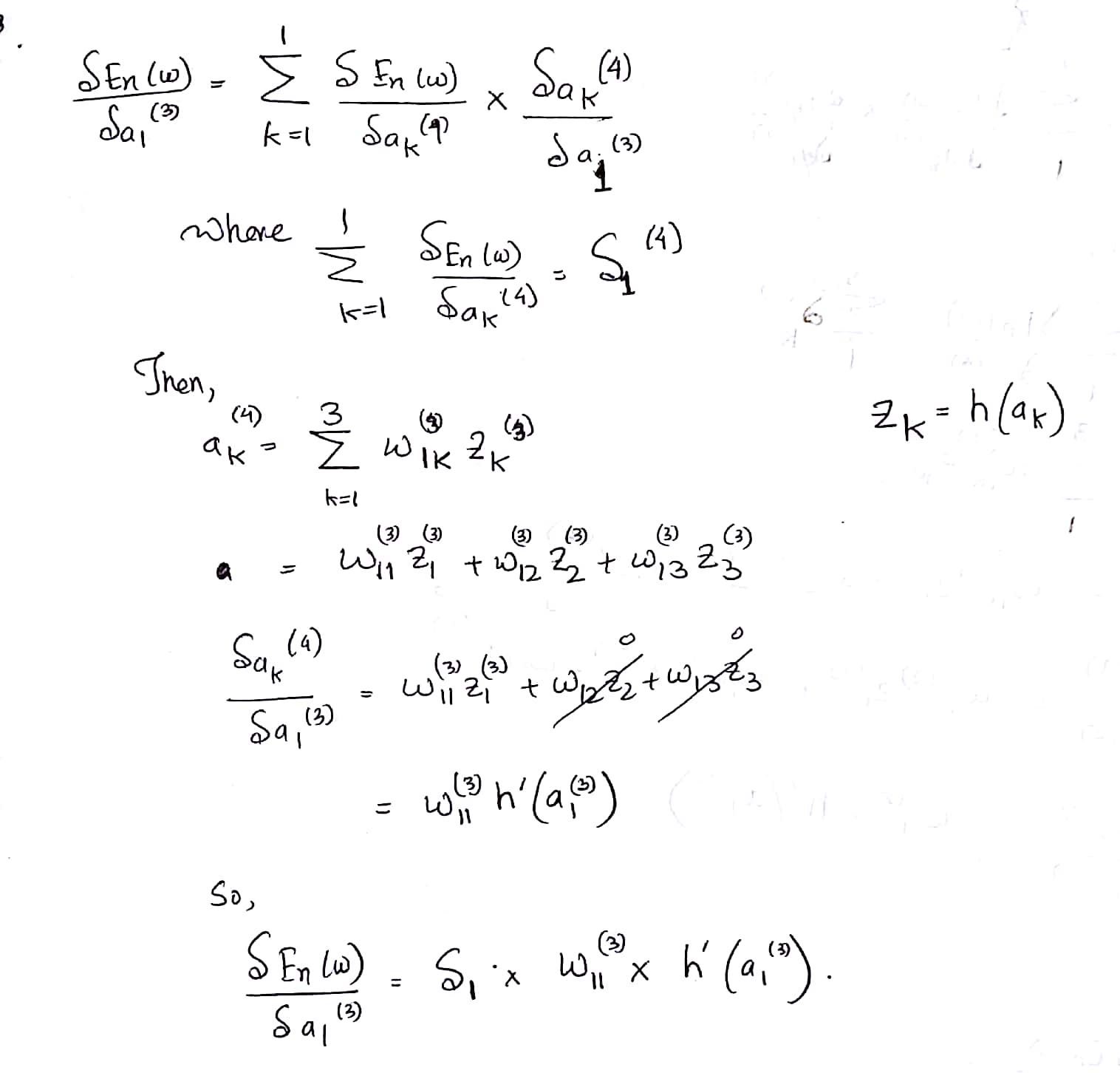
**Error Back Propagation**



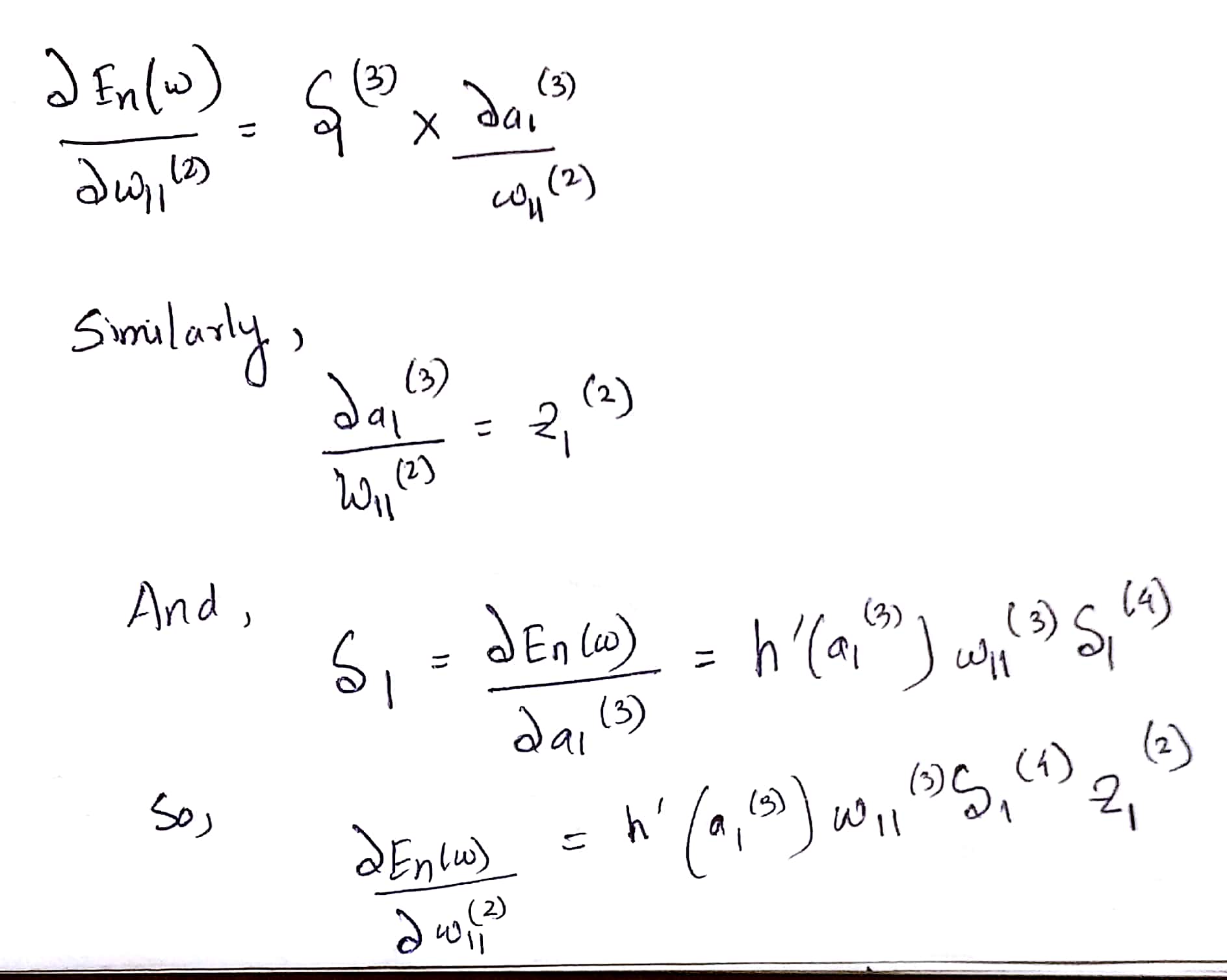
2.

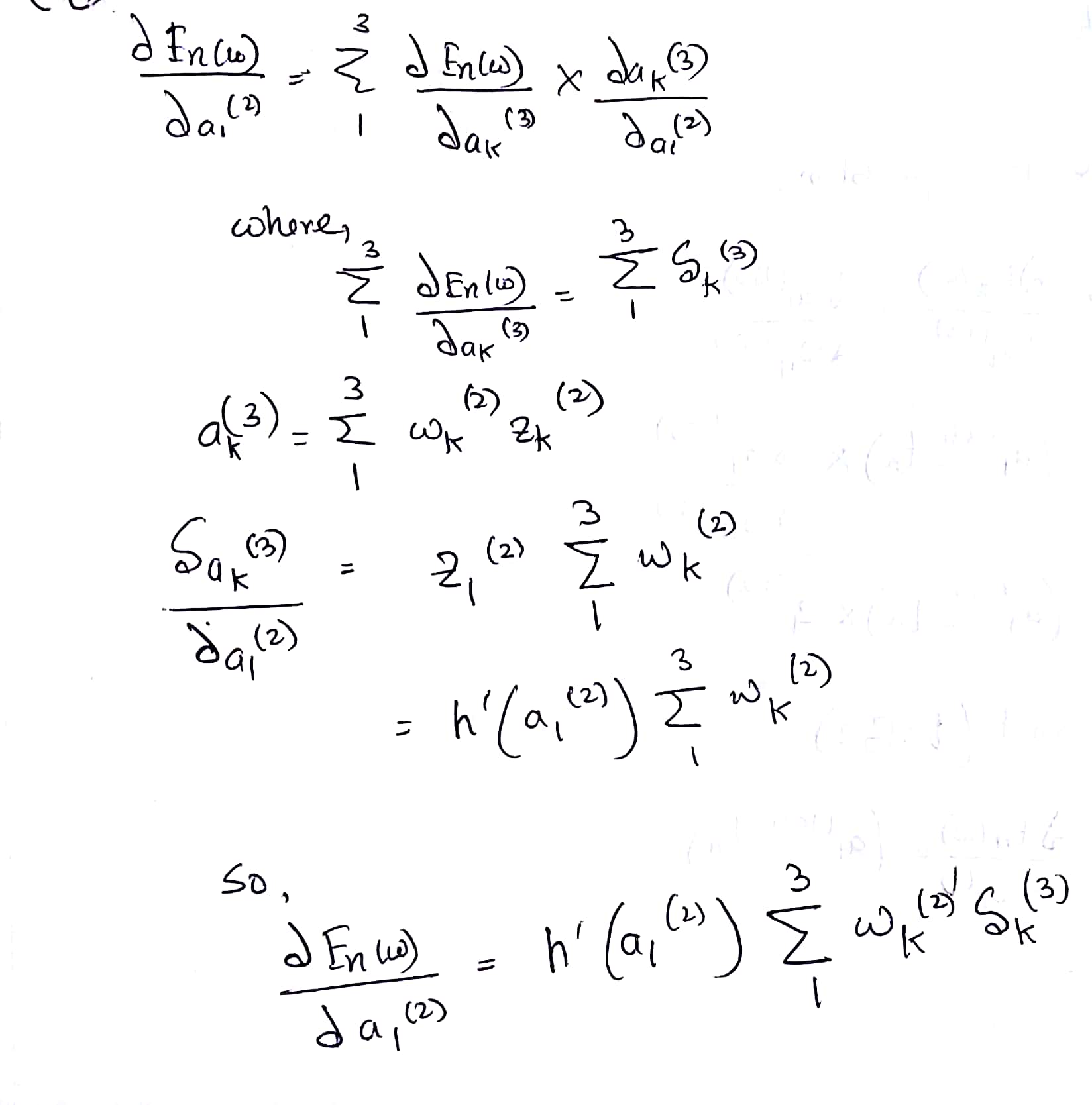
3.



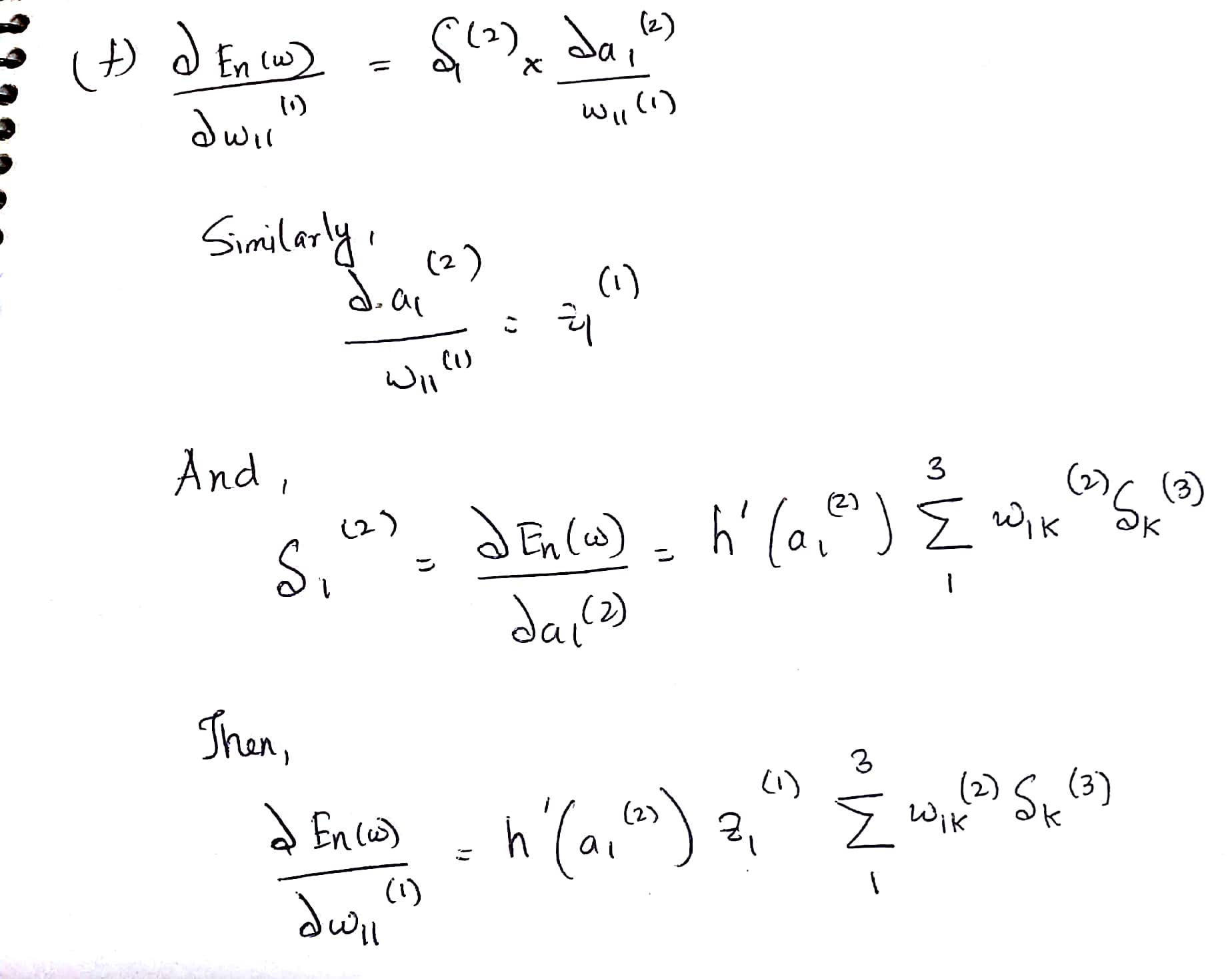
4.



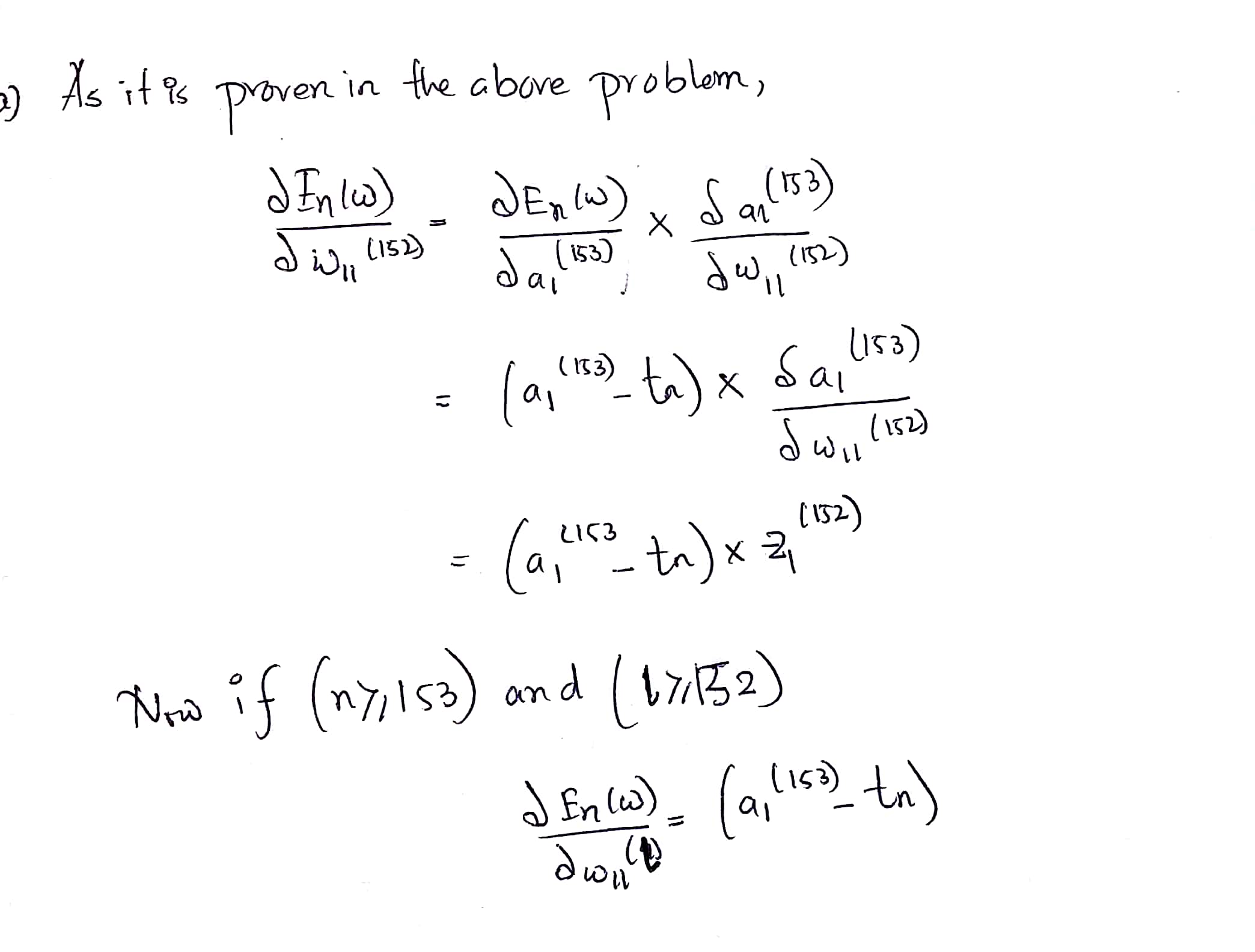
5.

****

6.

****

**Vanishing Gradient**



1. When training a deep model using back propagation, we calculate the gradient of the output with respect to the weight which is subtracted from those respective weight to make values more accurate to give correct output. The gradient tends to get smaller as we move backward in the network. The neuron in the earlier layers learn very slowly as compared to the neurons in the later layers in the hierarchy. The sigmoid function “squeezes” any input value into an output range of (0,1) which is very useful for representation of probabilities and classification. It is observed, when the sigmoid function value is either too high or too low, the derivative becomes very small i.e. << 1. This causes vanishing gradients and poor learning for deep networks. This can occur when the weights of our networks are initialized poorly – with too-large negative and positive values. These too-large values saturate the input to the sigmoid and pushes the derivatives into the small valued regions. However, even if the weights are initialized properly, and the derivatives are sitting around the maximum with many layers there will still be a vanishing gradient problem.
2. The *ReLU* activation returns its argument x whenever it is greater than zero and returns 0 otherwise. Then the first derivative of *ReLU* is equal to 1 when x is greater than zero, but otherwise it is 0. The advantage of *ReLU* activation function is that there will be no degradation of the error. The derivative is zero when x < 0 and this makes certain weights killed off. The back propagated error can be cancelled out whenever there is a negative input into a given neuron and therefore the gradient will also fall to zero.
3. For the case of bipartite graph, for the layer *l* +1 layer, we need to consider two nodes. So, their gradient will become zero when both ***h’(a1(l+1))*** and ***h’(a2(l+2))*** becomes zero (sum of them, which is the derivative of the cost function w.r.t ***W11l*** goes to zero.