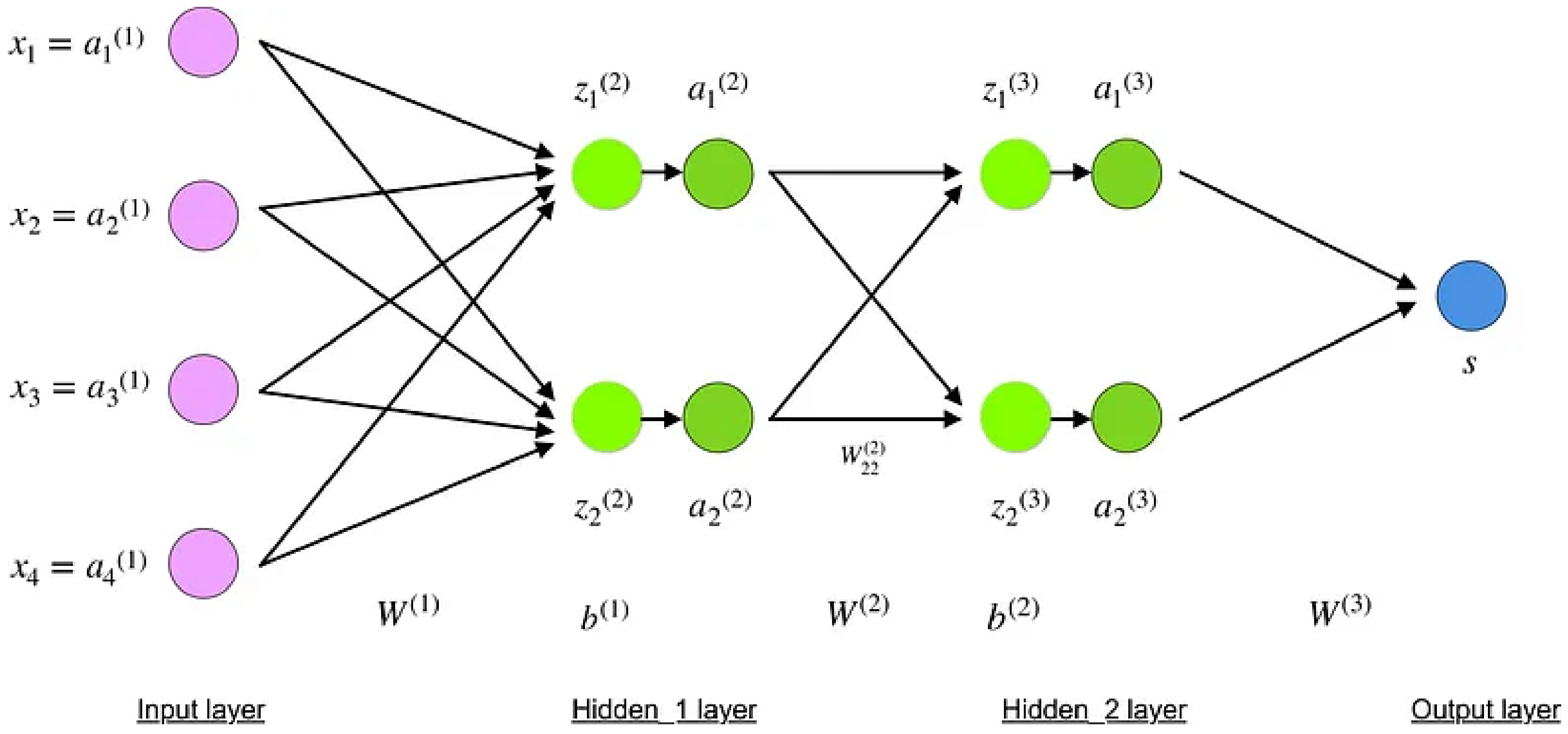
CS2109s Tutorial 8

by Lee Zong Xun

Recap

- **Backpropagation**
  - used to effectively train a neural network through a method called chain rule
  - after **each forward pass** through a network, backpropagation performs a backward pass while adjusting the model’s parameters

Let us go over the mathematical process of training and optimizing a simple 4-layer neural network.



Let’s pick layer 2 and its parameters as an example. The same operations can be applied to any layer in the network.

W^1 is a weight matrix of shape (n, m) where n is the number of output neurons (neurons in the next layer) and m is the number of input neurons (neurons in the previous layer). For us, n = 2 and m = 4.

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}$$

Note: In CS2109s, W^1 is a weight matrix of shape (4, 2) where 4 is the number of input neurons and 2 is the number of output neurons.

Highly recommended watch: https://www.youtube.com/watch?v=isPiE-DBagM&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6

Backpropagation lecture by Stanford NLP.

Question 1

Grace has a wine dataset which comprises of 1100 samples. Each wine sample has a label indicating which cultivar comes from, and two features: **colour intensity** and **alcohol level**.

100 of these samples are obtained from cultivar A and the remaining **1000** samples from cultivar B. Now, she wants to build a classifier that can predict which cultivar a wine sample comes from using the two features.

To deal with the imbalanced dataset, she decided to use the following loss function.

$$\mathcal{E} = -\frac{1}{n}\sum_{i=0}^{n-1}\left\{\alpha[Y_{(0,i)} \cdot log(\hat{Y}_{(0,i)})] + \beta[(1 - Y_{(0,i)}) \cdot log(1 - \hat{Y}_{(0,i)})]\right\}$$

where $\hat{Y} \in \mathbf{R}^{1\times n}$ and $Y \in \{0,1\}^{1\times n}$ such that $Y_{0,i} = 1$ if the ith wine sample is from cultivar A and $Y_{0,i} = 0$ if it is from cultivar B, and n is the number of wine samples (i.e. $n = 1100$ in this case).

To illustrate, we calculate the loss function for the following sample of 2:

$$Y = [0\ 1]$$

$$\hat{Y} = [0.2\ 0.9]$$

Loss function calculation:

$$\begin{aligned}\mathcal{E} &= -\frac{1}{2}\Big\{\beta[(1-0)\cdot log(1-0.2)] + \alpha[1\cdot log(0.9)]\Big\}\\ &= -\frac{1}{2}\Big\{\beta\cdot log(0.8) + \alpha\cdot log(0.9)\Big\}\end{aligned}$$

Question 1

Why do you think that she introduced the hyper-parameters α and β? How should she set their values?

Question 1

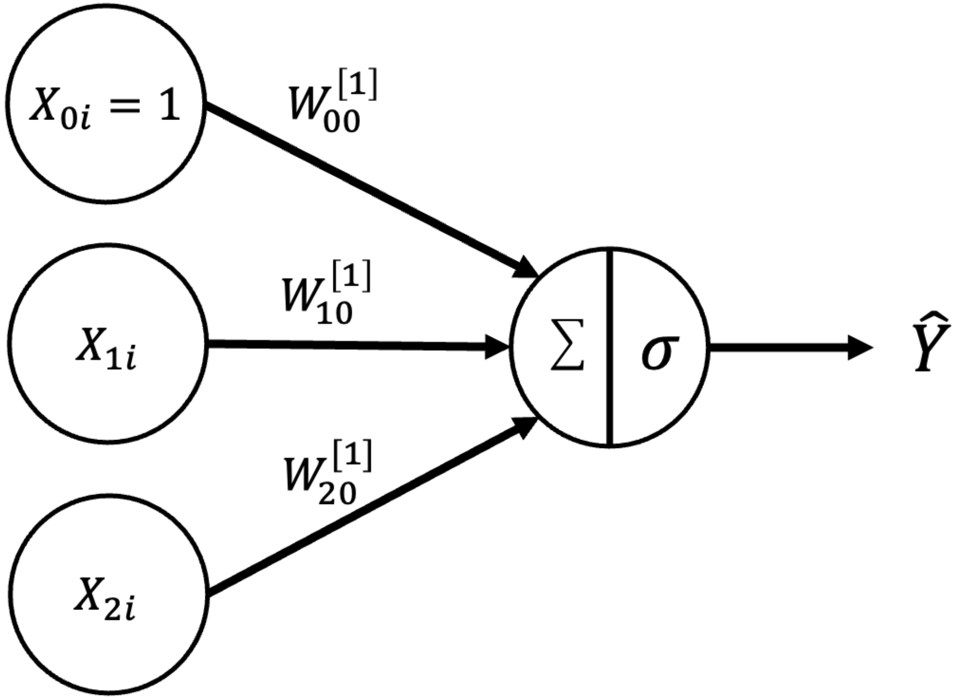
Why do you think that she introduced the hyper-parameters α and β? How should she set their values?

The hyper-parameters act as weights that determine how much each class contributes to the loss function. By setting $\alpha > \beta$, we can mitigate the problem of having a highly unbalanced dataset.

Note: There are multiple possible answers for α and β, but generally, their relationship should be $\alpha \approx 10\beta$.

Question 1

She decided to train a neural network with the architecture shown in the figure below.



Mathematically, this is given by

$$f^{[1]} = W^{[1]^T} X$$

$$\hat{Y} = g^{[1]}(f^{[1]})$$

where $W^{[1]} = [W_{00}^{[1]}\ W_{10}^{[1]}\ W_{20}^{[1]}]^T$, $X \in \mathbf{R}^{3\times n}$, **and** $g^{[1]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$.

In this question, for simplicity, let us consider the case where $n = 1$. **Show that:**

(i)

$$\frac{\partial \mathcal{E}}{\partial \hat{Y}} = \left[-\frac{\alpha Y_{00}}{\hat{Y}_{00}} + \frac{\beta(1 - Y_{00})}{1 - \hat{Y}_{00}} \right]$$

Note:

- $\frac{\partial \mathcal{E}}{\partial \hat{Y}} = \left[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \right]$ when $n = 1$.
- Y and $\hat{Y}$ are "effectively" scalars i.e. 1×1 matrix.

(ii) $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \left[- \frac{\alpha Y_{00}}{\hat{Y}_{00}} + \frac{\beta(1-Y_{00})}{1-\hat{Y}_{00}} \right] \left[\sigma(f_{00}^{[1]}) \Big(1 - \sigma(f_{00}^{[1]})\Big) \right]$

Note:

- f is "effectively" scalar.

(iii) $\frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}} = \left(\frac{\partial \mathcal{E}}{\partial f^{[1]}}\right)_{00} X_{20}$

Questions to note:

- What are we differentiating with respect to?

Question 1

Using your answer in (b), derive an expression for $\frac{\partial \mathcal{E}}{\partial W^{[1]}}$.

Observe that our solution in (b) also applies to other $\frac{\partial \mathcal{E}}{\partial W_{i0}^{[1]}}$, where $0 \leq i \leq 2$.

Furthermore,

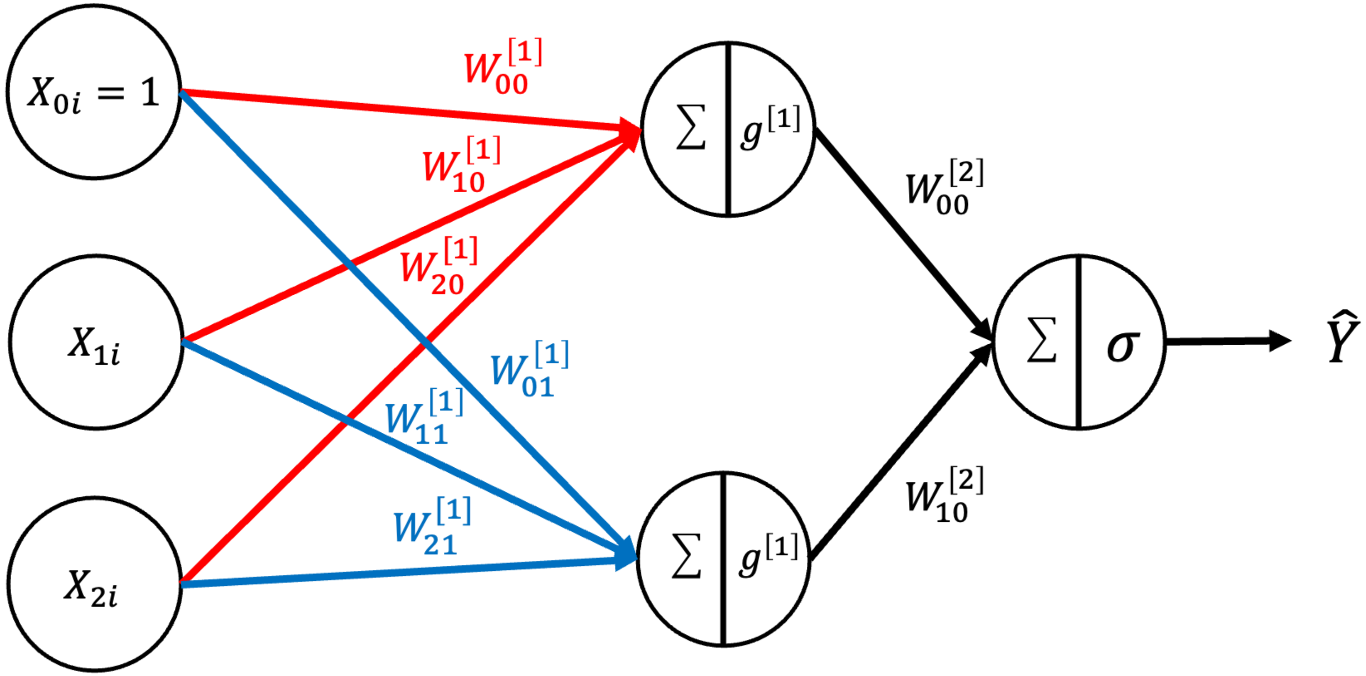
$$\frac{\partial \mathcal{E}}{\partial W^{[1]}} = \left[\frac{\partial \mathcal{E}}{\partial W_{00}^{[1]}} \frac{\partial \mathcal{E}}{\partial W_{10}^{[1]}} \frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}} \right]^T$$

Hence, we can conclude that

$$\frac{\partial \mathcal{E}}{\partial W^{[1]}} = \left(\frac{\partial \mathcal{E}}{\partial f^{[1]}} \right)_{00} X$$

Question 2

After training the neural network described in question 1, Grace observed that the training error is high. Thus, she decided to introduce a hidden layer, resulting in the architecture shown in the figure below



Mathematically, the network is given by

$$f^{[1]} = W^{[1]^T} X$$

$$a^{[1]} = g^{[1]}(f^{[1]})$$

$$f^{[2]} = W^{[2]^T} a^{[1]}$$

$$\hat{Y} = g^{[2]}(f^{[2]})$$

where $g^{[1]}(s) = ReLU(s)$, $g^{[2]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$, $W^{[1]} \in \mathbb{R}^{3\times 2}$ and $W^{[2]} \in \mathbb{R}^{2\times 1}$.

Similar to question 1, let us keep things simple and consider what happens when $n = 1$. In addition, assume that the loss function used remains the same.

Compute $\frac{\partial \mathcal{E}}{\partial W_{11}^{[1]}}$

Based on our previous observations in question 1, and the fact that the only entry in $f^{[1]}$ that is dependent on W_{11} is its (1, 0) entry, we get

$$\frac{\partial \mathcal{E}}{\partial W_{11}^{[1]}} = \frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[2]}} \frac{\partial f_{00}^{[2]}}{\partial a_{10}^{[1]}} \frac{\partial a_{10}^{[1]}}{\partial f_{10}^{[1]}} \frac{\partial f_{10}^{[1]}}{\partial W_{11}^{[1]}}$$

$$\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} = -\frac{\alpha Y_{00}}{\hat{Y}_{00}} + \frac{\beta(1 - Y_{00})}{1 - \hat{Y}_{00}}$$

$$\frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[2]}} = \sigma(f_{00}^{[2]})\left(1 - \sigma(f_{00}^{[2]})\right)$$

$$\frac{\partial f_{00}^{[2]}}{\partial a_{10}^{[1]}} = W_{10}^{[2]}$$

$$\frac{\partial a_{10}^{[1]}}{\partial f_{10}^{[1]}} = \begin{cases} 0, \text{if } f_{10}^{[1]} \leq 0 \\ 1, \text{otherwise} \end{cases}$$

$$= 1_{f_{10}^{[1]} > 0}$$

where $1_{f_{10}^{[1]} > 0}$ is an indicator function which makes notation simpler, especially for the final expression which we are supposed to derive.

Therefore,

$$\frac{\partial \mathcal{E}}{\partial W_{11}^{[1]}} = \Big[-\frac{\alpha Y_{00}}{\hat{Y}_{00}} + \frac{\beta(1 - Y_{00})}{1 - \hat{Y}_{00}} \Big] \sigma(f_{00}^{[2]}) \Big(1 - \sigma(f_{00}^{[2]})\Big) W_{10}^{[2]} 1_{f_{10}^{[1]} > 0} X_{10}$$

Question 3

Let us consider a generic neural network with the following architecture.

$$f^{[1]} = W^{[1]^T} X$$

$$\hat{Y} = g^{[1]}(f^{[1]})$$

and with the following loss function

$$\mathcal{E} = -\frac{1}{n} \sum_{i=0}^{n-1} \left\{ [Y_{0,i} \cdot log(\hat{Y}_{0,i})] + [(1 - Y_{0,i}) log(1 - \hat{Y}_{0,i})] \right\}$$

In particular, let $n \in \mathbb{N}$, $X \in \mathbb{R}^{m_0 \times n}$, $\hat{Y} \in \mathbb{R}^{1 \times n}$, $Y \in \{0, 1\}^{1 \times n}$, $W^{[1]} \in \mathbb{R}^{m_0 \times 1}$ and m_0 refers to the number of features.

Show that $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \hat{Y} - Y$ when $n = 1$.

Notice that $\alpha = \beta = 1$. Using this piece of information, we will thus have

$$\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} = -\frac{Y_{00}}{\hat{Y}_{00}} + \frac{(1 - Y_{00})}{1 - \hat{Y}_{00}}$$

Moreover, from question 1(b), we also have

$$\frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[1]}} = \sigma(f_{00}^{[1]})\left(1 - \sigma(f_{00}^{[1]})\right)$$

However, now, it is helpful for us to simplify this equation. Notice that

$$\hat{Y}_{00} = \sigma(f_{00}^{[1]})$$

Thus,

$$\frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[1]}} = \hat{Y}_{00}(1 - \hat{Y}_{00})$$

Then,

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial f^{[1]}} &= \left[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f_{00}^{[1]}} \right] \\ &= \left[\hat{Y}_{00} - Y_{00} \right] \\ &= \hat{Y} - Y \end{aligned}$$

Using your answer to (a), find $\frac{\partial \mathcal{E}}{\partial f^{[1]}}$ when $n \in \mathbb{N}$.

Solution:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial f^{[1]}} &= \frac{1}{n}\left[(\hat{Y}_{00} - Y_{00})\ (\hat{Y}_{01} - Y_{01})\ \dots\ (\hat{Y}_{0n} - Y_{0n})\right] \\ &= \frac{1}{n}(\hat{Y} - Y)\end{aligned}$$

Note that the $\frac{1}{n}$ comes from the loss function.

Question 4

Suppose we use σ (the sigmoid function) as our activation function in a neural network with 50 hidden layers.

- Play around with the code. Notice that when performing back propagation, the gradient magnitudes of the first few layers are extremely small. What do you think causes this problem?
- Based on what we have learnt thus far, how can we **mitigate** this problem?

Solution:

This problem is known as *vanishing gradient*.

- Observe that to compute $\frac{\partial \mathcal{E}}{\partial W^{[1]}}$, we need to take a product of many derivatives.
- The derivative of σ returns a value that is certainly between 0 and 0.25.
- What happens if we multiply many of such values together?
    - We will end up with a very small number!
    - Consequently, change in weights may be very small, causing convergence to be slow, if not impossible.
    - Sounds familiar? This is exactly the scenario that most of you have encountered in earlier PSETs!

A possible mitigation is to use the **ReLU function** (or its variants).