National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester I, 2023/2024

Tutorial 8
Back Propagation

These questions will be discussed during the tutorial session. Please be prepared to answer them.

Summary of Key Concepts

In this tutorial, we will discuss and explore the following key learning points/lessons from lecture:

1. Back propagation
2. Matrix calculus for back propagation
3. Potential issues with back propagation

Warning: Lots of math ahead! For "Show that" questions, start from the *left hand side* of the equation

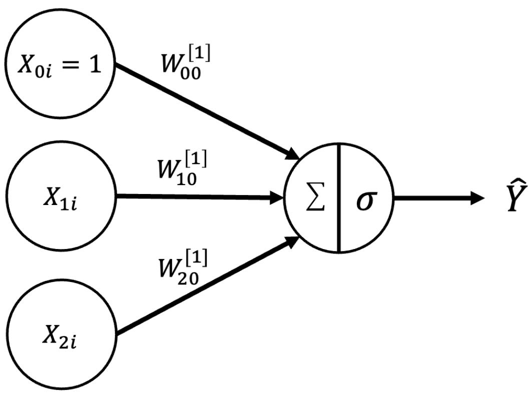
Problems

1. **Back Propagation (Warm Up)**

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

   Now, she wants to build a classifier that can predict which cultivar a wine sample comes from using the two features.

   (a) 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 $W^{[1]} = [W^{[1]}_{00}\ W^{[1]}_{10}\ W^{[1]}_{20}]^T$, $X \in \mathbf{R}^{3\times n}$, and $g^{[1]}(s) = \sigma(s) = \frac{1}{1+e^{-s}}$. Note that here, $X_{0i} = 1$, and X_{1i} and X_{2i} represent the colour intensity and alcohol level respectively.

She decided to use the following loss function[1]

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

where $\hat{Y} \in \mathbf{R}^{1\times n}$ and $Y \in \{0, 1\}^{1\times n}$ such that $Y_{0i} = 1$ if the ith wine sample is from cultivar A and $Y_{0i} = 0$ 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:

$$\mathcal{E} = -\frac{1}{2}\Big[(1 - 0) \cdot log(1 - 0.2) + 1 \cdot log(0.9)\Big]$$
$$= -\frac{1}{2}\Big[log(0.8) + log(0.9)\Big] = 0.16425$$

Note that $\log(1) = 0$ and $\log(0) \to -\infty$. We assume that $\log(x) = \ln(x)$ in this tutorial.

For parts (a) and (b), to keep things simple, let us consider the case where $n = 1$. **Show that:**

i. $\frac{\partial\mathcal{E}}{\partial\hat{Y}} = \Big[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1-Y_{00}}{1-\hat{Y}_{00}}\Big]$

We provide the answer for this part as a guiding example:

Note that $\frac{\partial\mathcal{E}}{\partial\hat{Y}} = \Big[\frac{\partial\mathcal{E}}{\partial\hat{Y}_{00}}\Big]$ when $n = 1$

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

Therefore,

$$\frac{\partial\mathcal{E}}{\partial\hat{Y}} = \Big[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1 - Y_{00}}{1 - \hat{Y}_{00}}\Big]$$

Try to plug in the case where (1) the true label $Y_{00} = 0$ and (2) $Y_{00} = 1$. This equation tells us how the change in predicted value $\hat{Y}$ will be able to influence the change of error/loss value.

[1] This loss function is usually known as *log loss* or *binary cross-entropy*.

ii. $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \hat{Y} - Y$

Hint: Since $n = 1$, $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial f^{[1]}_{00}} \end{bmatrix}$. By chain rule, $\frac{\partial \mathcal{E}}{\partial f^{[1]}_{00}} = \frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f^{[1]}_{00}} \cdots$

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

Hint: Since $n = 1$, $f^{[1]} = \begin{bmatrix} f^{[1]}_{00} \end{bmatrix}$. Thus, $\frac{\partial f^{[1]}}{\partial W^{[1]}_{20}} = \begin{bmatrix} \frac{\partial f^{[1]}_{00}}{\partial W^{[1]}_{20}} \end{bmatrix} \cdots$

NOTE: $\left(\frac{\partial \mathcal{E}}{\partial f^{[1]}}\right)_{00}$ refers to the (0, 0) entry of the matrix $\frac{\partial \mathcal{E}}{\partial f^{[1]}}$.

You should use the chain rule to derive your answers. Treat $f^{[1]}$, $W^{[1]}$, $\hat{Y}$, Y and X as matrices.

Solution:

ii. Since $n = 1$, $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial f^{[1]}_{00}} \end{bmatrix}$. By chain rule, $\frac{\partial \mathcal{E}}{\partial f^{[1]}_{00}} = \frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f^{[1]}_{00}}$.
Now, let us compute $\frac{\partial \hat{Y}_{00}}{\partial f^{[1]}_{00}}$.

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

It is helpful for us to simplify this equation. Notice that

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

Thus,

$$\begin{aligned} \frac{\partial \hat{Y}_{00}}{\partial f^{[1]}_{00}} &= \hat{Y}_{00}(1 - \hat{Y}_{00}) \\ \frac{\partial \hat{Y}}{\partial f^{[1]}} &= \Big[\hat{Y}_{00}(1 - \hat{Y}_{00})\Big] \end{aligned}$$

Then,

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial f^{[1]}} &= \Big[\frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f^{[1]}_{00}}\Big] \\ &= \Big[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1 - Y_{00}}{1 - \hat{Y}_{00}}\Big]\Big[\hat{Y}_{00}(1 - \hat{Y}_{00})\Big] \\ &= \Big[-Y_{00}(1 - \hat{Y}_{00}) + (1 - Y_{00})\hat{Y}_{00}\Big] \\ &= \Big[\hat{Y}_{00} - Y_{00}\Big] \\ &= \hat{Y} - Y. \end{aligned}$$

iii. Since $n = 1$, $f^{[1]} = \begin{bmatrix} f^{[1]}_{00} \end{bmatrix}$. Thus, $\frac{\partial f^{[1]}}{\partial W^{[1]}_{20}} = \begin{bmatrix} \frac{\partial f^{[1]}_{00}}{\partial W^{[1]}_{20}} \end{bmatrix}$. Moreover,

$$f^{[1]}_{00} = \sum_{i=0}^{2} (W^{[1]^T})_{0i} X_{i0} = \sum_{i=0}^{2} W^{[1]}_{i0} X_{i0}$$

As such,

$$\frac{\partial f^{[1]}_{00}}{\partial W^{[1]}_{20}} = X_{20}$$

Then, by chain rule, we get

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

To summarise, $\frac{\partial \mathcal{E}}{\partial \hat{Y}} = \left[-\frac{Y_{00}}{\hat{Y}_{00}} + \frac{1-Y_{00}}{1-\hat{Y}_{00}} \right]$, $\frac{\partial \mathcal{E}}{\partial f^{[1]}} = \frac{\partial \mathcal{E}}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial f^{[1]}}$, and $\frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}} = \left(\frac{\partial \mathcal{E}}{\partial f^{[1]}}\right)_{00} X_{20}$.

NOTE: $\frac{\partial \mathcal{E}}{\partial \hat{Y}}$, and $\frac{\partial \mathcal{E}}{\partial f^{[1]}}$ are matrices since $\mathcal{E}$ is a scalar, but $\hat{Y}$ and $f^{[1]}$ are matrices. However, $\frac{\partial \mathcal{E}}{\partial W_{20}^{[1]}}$ is a scalar since $W_{20}^{[1]}$ is a scalar.

(b) Using your answer in (a), derive an expression for $\frac{\partial \mathcal{E}}{\partial W^{[1]}}$. Recall that Back propagation is the act of nudging/fine-tuning the weights W during training to minimize the loss $\mathcal{E}$. Does your derived expression prove that Back propagation works?

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

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

This expression tells us that the change in first layer weighted sum $f^{[1]}$, will influence the change in predicted value $\hat{Y}$, which in turn will influence the change of loss $\mathcal{E}$. Meaning, we could **decrease the loss by changing the weights**. Especially if the previous neuron X is more active (larger), it will have stronger effect on the loss. This is the intuition behind Back propagation.

(c) Let us consider a general case where $n \in \mathbb{N}$. Using your answer to (a), find $\frac{\partial \mathcal{E}}{\partial f^{[1]}}$ when $n \in \mathbb{N}$. Check your answer using the Python notebook.

Hint: Read the Python notebook.

Solution:

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

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

(d) Let's say that Grace has 100 samples from cultivar A and 1000 samples from cultivar B that make up the 1100 total samples. To deal with the imbalanced data set, she decided to introduce two hyper-parameters α and β in the loss function, as shown below.

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

For example, using the same sample of 2 from (a), the loss function can be computed as follows:

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

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

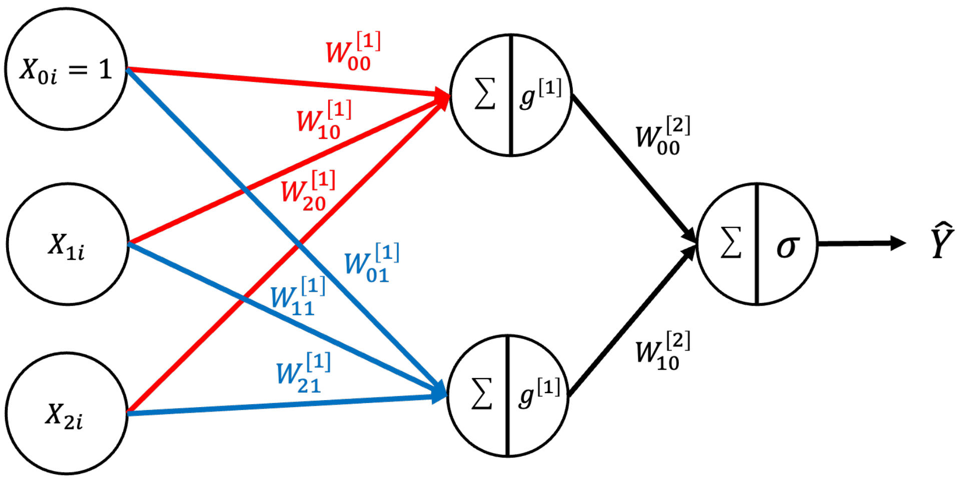
Solution: The hyper-parameters act as weights that determine how much each class contributes to the loss function. By setting $\alpha > \beta$, we can mitigate the problem of having a highly unbalanced dataset.

There are multiple possible answers for α and β, but generally, their relationship should be $\alpha \approx 10\beta$ (since the dataset has 10 times more samples from cultivar B than those from cultivar A).

In other words, we punish the machine more heavily if it misclassifies A. That way, the machine won't be biased towards predicting all samples as B.

2. **Back Propagation for a Deep(er) Network**

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 $W^{[2]} \in \mathbb{R}^{2\times 1}$.

The $ReLU$ function is defined as follows:

$$ReLU(s) = \begin{cases} s & \text{if } s > 0 \\ 0 & \text{otherwise} \end{cases}$$

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 $\frac{\partial \mathcal{E}}{\partial W^{[1]}_{11}}$.

Recall again that in Back propagation, we can decrease or increase the loss by changing the weights. But how strong the effect of weight nudging on the loss also depends on the neuron X associated with the weight. In this question, we introduce hidden layer that has its own neurons activation. Based on your derived expression, how can this hidden layer amplify the effect of weight nudging that helps minimize the training error?

Your answer should not contain any partial derivatives on the right hand side, they should all be evaluated.

Hint: The results found/observations made in question 1 are likely to be useful here.

Solution: 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

$$\frac{\partial \mathcal{E}}{\partial W^{[1]}_{11}} = \frac{\partial \mathcal{E}}{\partial \hat{Y}_{00}} \frac{\partial \hat{Y}_{00}}{\partial f^{[2]}_{00}} \frac{\partial f^{[2]}_{00}}{\partial a^{[1]}_{10}} \frac{\partial a^{[1]}_{10}}{\partial f^{[1]}_{10}} \frac{\partial f^{[1]}_{10}}{\partial W^{[1]}_{11}}$$

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

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

$$\frac{\partial f^{[2]}_{00}}{\partial a^{[1]}_{10}} = W^{[2]}_{10}$$

$$\begin{aligned}\frac{\partial a^{[1]}_{10}}{\partial f^{[1]}_{10}} &= \begin{cases} 0, \text{if } f^{[1]}_{10} \leq 0 \\ 1, \text{otherwise} \end{cases} \\ &= 1_{f^{[1]}_{10} > 0}\end{aligned}$$

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

$$\frac{\partial f^{[1]}_{10}}{\partial W^{[1]}_{11}} = X_{10}$$

Therefore,

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

Notice that in the first question with no hidden layer, we cannot modify neurons X, since they are fixed input data given. Hence, the effect of weight nudging on reducing training error is limited.

However, we can modify the neurons activations in the hidden layers to be more or less active, and we do this by nudging the weights in the previous layers $W^{[1]}$. Thus, we can amplify the effect of weight nudging in the next layer $W^{[2]}$ to further minimize the training error.

3. **Potential Issues with Training Deep Neural Networks**

Suppose we use σ (the sigmoid function) as our activation function in a neural network with 50 hidden layers, as per the code in the accompanying Python notebook.

(a) 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?

(b) Based on what we have learnt thus far, how can we **mitigate** this problem? Test out your solution by modifying the code and checking the gradient magnitudes.

Hint: You don't need to change much.

Solution: This problem is known as *vanishing gradient.*

Referring to previous question, observe that to compute $\frac{\partial \mathcal{E}}{\partial W^{[1]}}$, we need to take a product of many derivatives. Moreover, recall that the derivative of σ returns a value that is certainly between 0 and 0.25. Therefore, if we multiply many of such values together, we will end up with a very small number. Consequently, our change in weights for each iteration may be very small, causing convergence to be slow, if not impossible.

A possible mitigation is to use the ReLU function (or its variants). As long as the input into the ReLU function is non-negative, its derivative will be 1, thereby precluding the aforementioned issue.