深度學習

HW5

學號 : B103012002

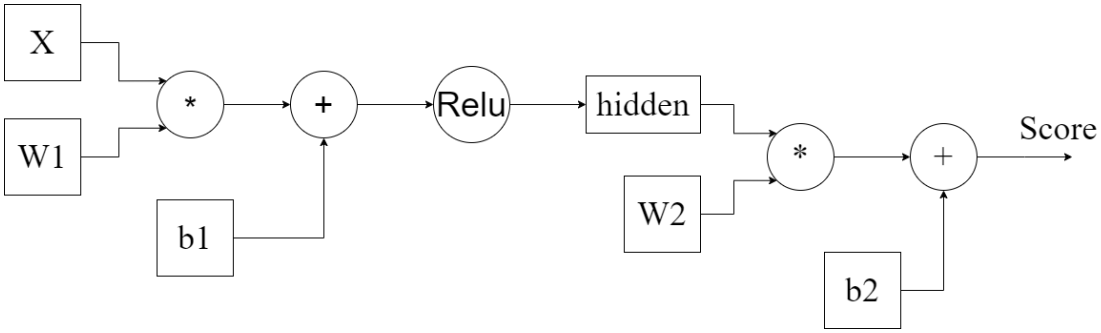
姓名 : 林凡皓

一、 Forward pass

- nn_forward_pass :

1. 解題思路 :

   本題主要是計算神經網路對於每個類別計算出來的分數，整個神經網路的 forward path 如下圖



   根據上圖，先利用 torch.clamp( )來計算經過 relu 後的 hidden。接著透過 torch.mm( )來計算最後的 score。

2. 執行結果 :

```
Your scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')
torch.float32

correct scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')

Difference between your scores and correct scores: 2.24e-11
```

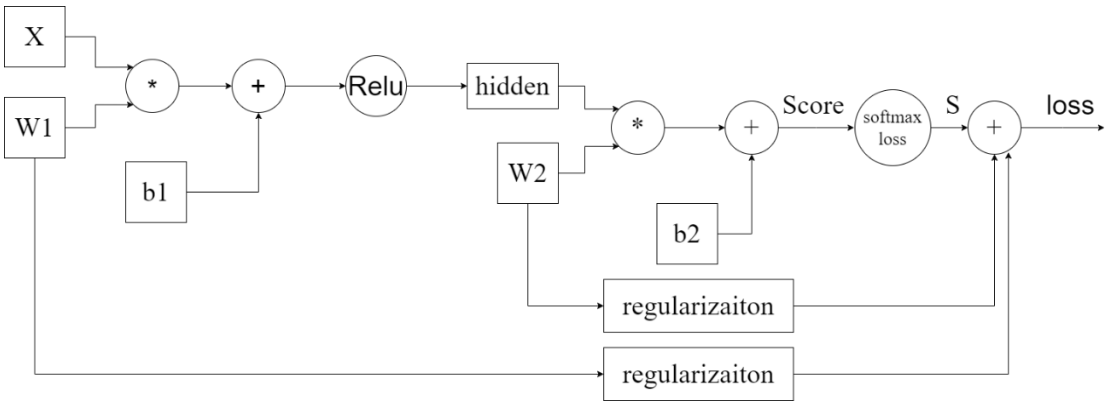
3. 額外討論 :

   經過 relu 後，小於 0 的數字會被歸零，因此使用 torch.clamp( )可以很好的去達到這樣的效果。torch.clamp( )的主要功能為限制 tensor 的範圍，使用的語法為 torch.clamp(input, min, max)，其中 min 為 tensor 的最小值，max 為 tensor 的最大值。

- nn_forward_backward :

1. 解題思路 :

本題主要是利用 forward propagation 的方式來計算 loss。延伸 nn_forward_path 時的神經網路架構，如下圖



根據 softmax loss 的定義，如下圖，來完成一個 function。

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

在這題，由於網路架構有兩層，因此會有兩個權重值 tensor 分別為 W1 和 W2，因此 regularization term 會有兩項分別為 R(W1)和 R(W2)。

2. 執行結果 :

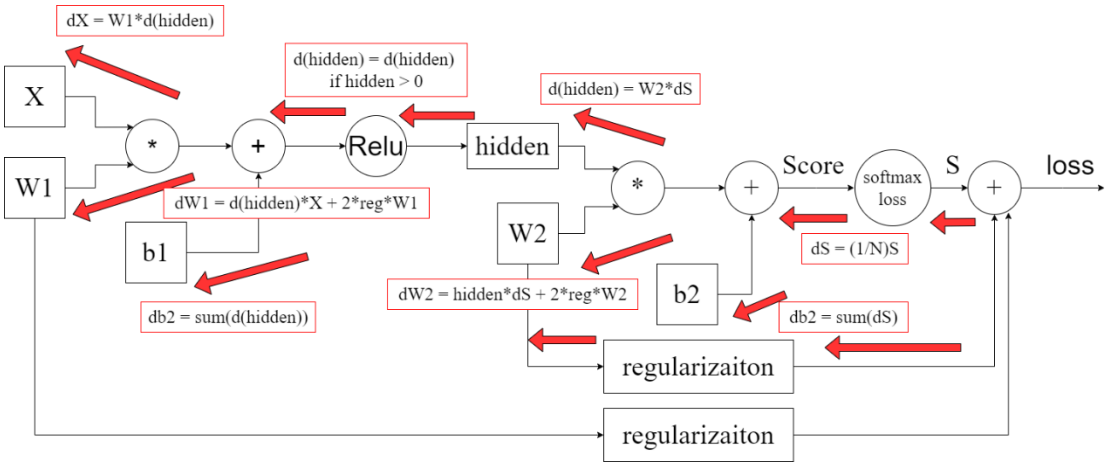
```
Your loss:  1.0986121892929077
Correct loss:  1.0986121892929077
Difference: 0.0000e+00
```

二、 Backward pass

- nn_forward_backward :

1. 解題思路 :

使用 computation graph 來實作 back propagation，computational graph 與計算之 gradient 如下



利用此計算結果加上矩陣相乘的規則(前面矩陣的 column 數目要等於後面矩陣的 row 數目)來判斷哪個矩陣要放前面，或是誰需要做轉置，來完成 back propagation。

2. 執行結果 ：

```
W2 max relative error: 1.261262e-06
b2 max relative error: 3.122771e-09
b1 max relative error: 8.239303e-06
W1 max relative error: 1.441750e-06
```

三、 Train the network

- nn_train :

1. 解題思路 ：

   Trianing 的主要在做的事就是更新權重值。更新權重的方式是 gradient descent，也就是往負的 gradient 方向去走。因此，寫法就是 w = w – learning_rate * dw。

2. 執行結果 ：

```
Final training loss:  0.5211756229400635
```

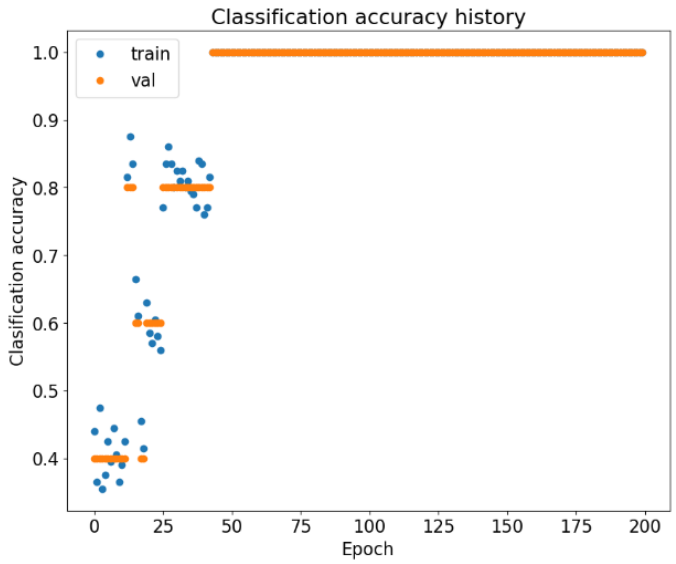
由上圖可以看出，因為 gradient descent 的影響，training loss 會隨著訓練的次數增加而下降，直到最後趨近於飽和，代表說模型的極限已到。

- nn_predict :

1. 解題思路 :

   nn_predict 是用來當作 nn_train 的 pred_func 參數。將我們選擇好的參數、training data 送進我們選擇的計算分數的 function 中，接著在計算的分數中找到最大值的 index 即為預測結果。

2. 執行結果 :



由上圖可以看出準確度隨訓練次數增加而上升，最後趨近飽和，

趨勢有點類似於 loss。

3. 額外討論 :

為甚麼 loss function 可以用來計算分數?

關於這個問題，先看到 nn_predict 傳入的 loss_func。這裡的 loss_func 使用的是先前定義好的 nn_forward_backward。在 nn_forward_backward 中可以看到有一段 code 是使用 nn_forward_pass 來計算分數，並在 y = None 時，nn_forward_backward 會回傳 score 而不是 loss 和 grad。接著看到定義 nn_pridect 的地方。在定義 nn_predict 的時候，我們並沒有傳入 y，也就是說 y 是 None，因此 nn_forward_backward，就會變成是一個計算分數的函數。

四、 額外嘗試

- Gradient problem :

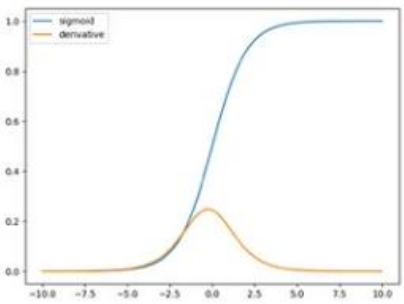
Gradient descent 為訓練模型中很重要的一環，在訓練神經網路時，有兩個常見的 gradient 問題，分別為 gradient vanishing 和 gradient exploding。接下來討論這兩個問題

1. Gradient vanishing :

$$\frac{\partial L}{\partial w_{11}^{1}} = \frac{1}{n}(\hat{y} - y)[\sum_{i}^{2} w_{1i}^{3}\sigma'(z_i^2)w_{i1}^{2}]\sigma'(z_1^1)x_1$$

1. 2. 3.

1. $(\hat{y} - y)$: Defined by your loss function .
2. w : Defined by your initialization weight .
3. $\sigma'(z)$: Defined by your activation function .



由上圖可以看到，可能影響 gradient 的因素有三個，分別為(預測值 – 標籤值)、權重、activation function 微分。這三者中，我認為(預測值 – 標籤值)是不會對梯度造成影響的，因此會針對權重與 activation function 做討論。

- 權重 W : 權重的初始化確實可能造成訓練上的問題，但是初始化的問題容易解決，不論是使用多次不一樣的初始化然後做平均，或是使用一些初始化策略像是 Normal、uniform、truncated_norm 等，都可以有效解決初始化的問題。

- Activation function : activation function 的微分會是造成梯度消失的主要原因。sigmoid function 就有發生梯度消失的可能，可以在上圖的右下角處看到，sigmoid 微分的最大值大約落在 0.25 左右而已。以三層的 NN 來說，gradient 縮小倍率是 0.25^2，也就是 0.0625 倍。光是三層的 NN 就有如此可觀的下降，更何況是更加大型的 NN。這也是為甚麼使用 sigmoid function 會發生梯度消失的問題。

了解原因後，梯度消失的解決方法如下

- 不要使用 sigmoid，改用其他 activation function 像是 relu，relu 微分後只有 0 或是 1，比起 sigmoid 不容易發生梯度消失的問題。
- 在 NN 中搭配 Batch normalization。
- NN 不要太深。

2. Gradient exploding :

梯度爆炸主要原因如下

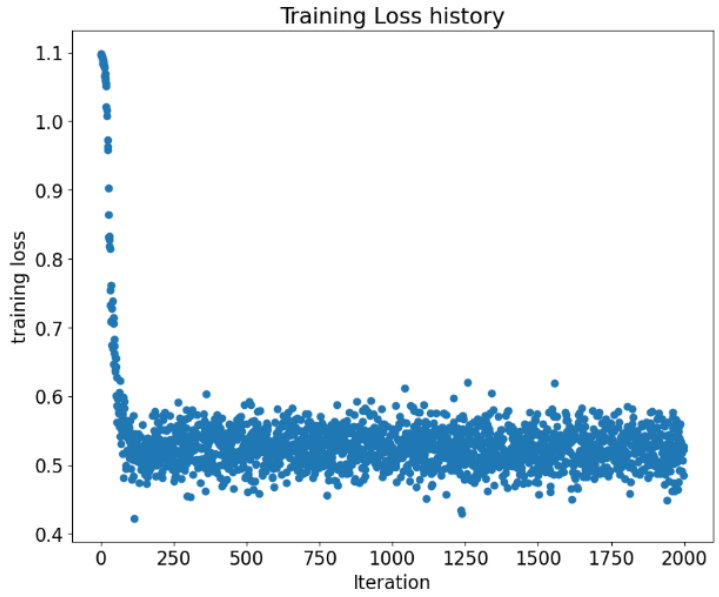
- 梯度累積 : 在 back propagation 中，如果梯度增長太快，可能會造成權重值更新太大。
- 權重初始化 : 如果權重的初始化沒做好，這會導致靠近輸入層的權重變化很大。

對於梯度爆炸，討論的人數其實不多，這是因為梯度爆炸問題很容易解決，像是設定說如果計算出來的梯度小於-1，就讓梯度等於-1，如果梯計算出來的梯度大於 1，就將梯度設為 1。

- 嘗試調整 training 參數 :

1. 調整 num_iters :
   - 將 num_iters 提升到 2000，結果如下

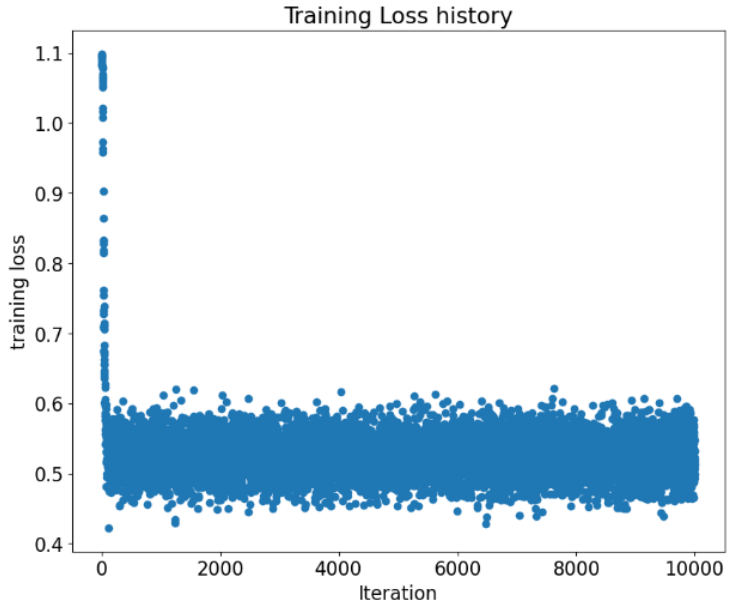
```
Final training loss:  0.4844719469547272
```



可以看到說，training loss 來到 0.48，比 num_iters = 200 時的 0.52 來的低一些，代表說訓練 200 次的模型經過更多次的訓練，仍然可以降低 loss。

- 將 num_iters 持續加大到 10000，結果如下

```
Final training loss:  0.5316468477249146
```



可以看到 training loss 不降反升，代表說 num_iters = 2000 時的結果已經是這個模型的極限，再繼續訓練下去只會適得其反。

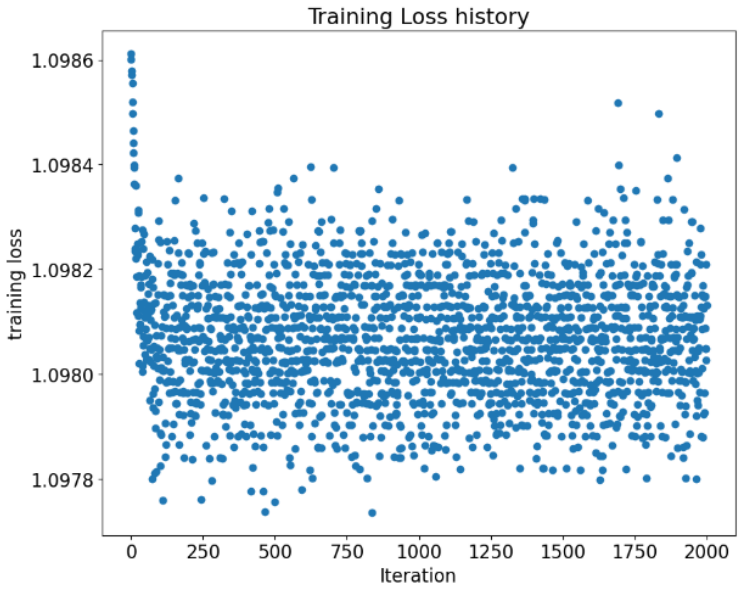
- 結論：以調整 num_iters 來降低模型的 loss 是可行的，但是嚴格來說，嘗試不同的 num_iters 並不是讓模型更加厲害，只

是在尋找對於目前模型來說，最佳的表現會發生在哪裡。如果找到目前模型的最佳表現後，還希望去降低 loss，那調整模型架構會是比較好的選擇。

2. 調整 learning rate :

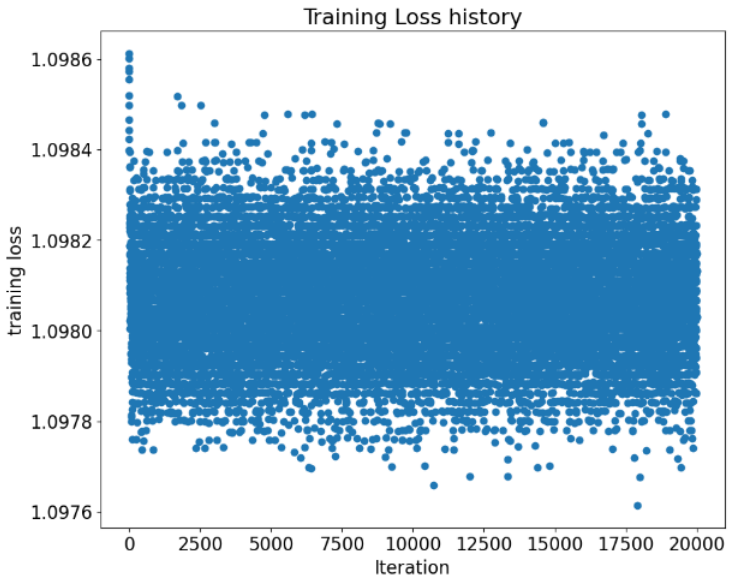
   ➢ 將 learning rate 降到 1e-3，結果如下

```
Final training loss:  1.0981323719024658
```



由上圖可以看出，loss 很高，代表說現在是 underfitting 的情況，因此常是加大訓練次數，結果如下

```
Final training loss:  1.0980288982391357
```

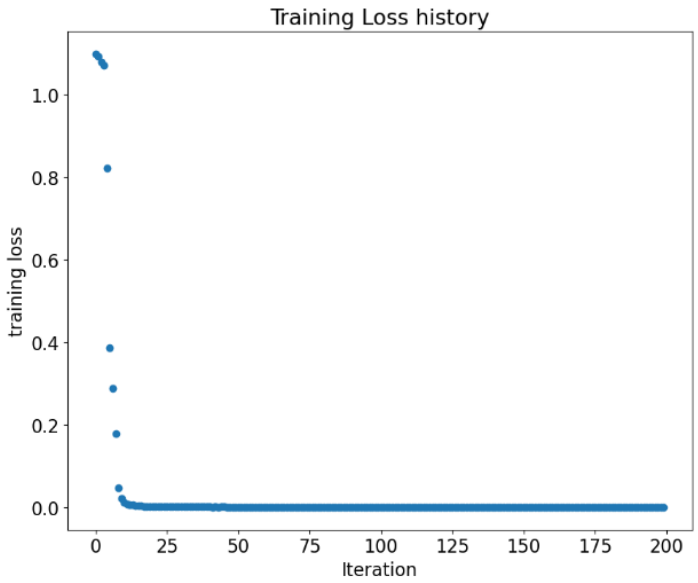


我們會發現到，loss 並沒有下降，代表說 training 的過程中，掉到了 local minimum 的地方，因為 learning rate 過小的關

係，導致無論如何我們都無法跳脫 local minimum 的地方，所以即便增加訓練次數，對於 loss 基本上沒有甚麼幫助。

- 由降低 learning rate 的經驗可以得知，在 training 的過程中有可能會掉到 loacal minimum 中，因此不免讓我懷疑在最一開始的模型(learning rate = 1e-1)是否也有掉入 local minimum 的現象，因此我加大 learning rate 到 learning rate = 1 來查看結果，結果如下

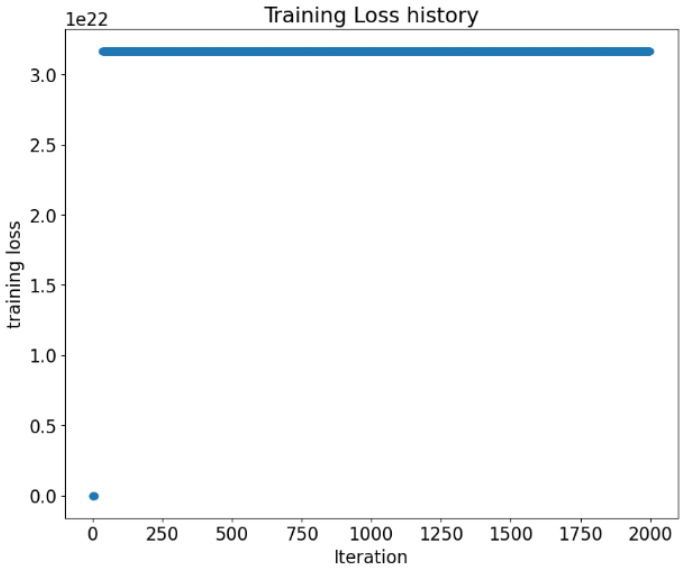
```
Final training loss:  0.0012291725724935532
```



可以看出，加大 learning rate 使 loss 下降很多，也就是說在最一開始的訓驗過程中有掉入 local minimum 的現象。

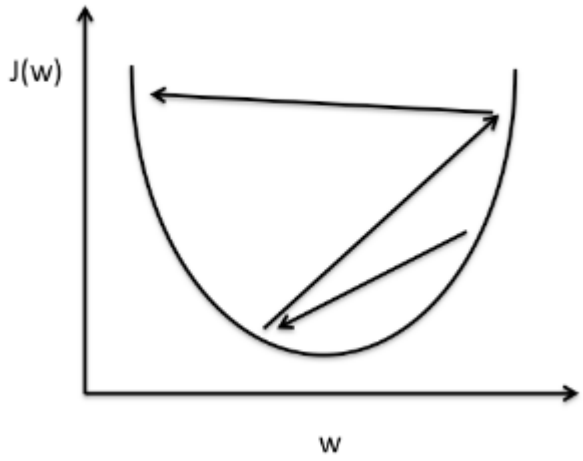
- 持續加大 learning rate 到 learning rate = 1.8，結果如下

```
Final training loss:  3.1659816739855017e+22
```



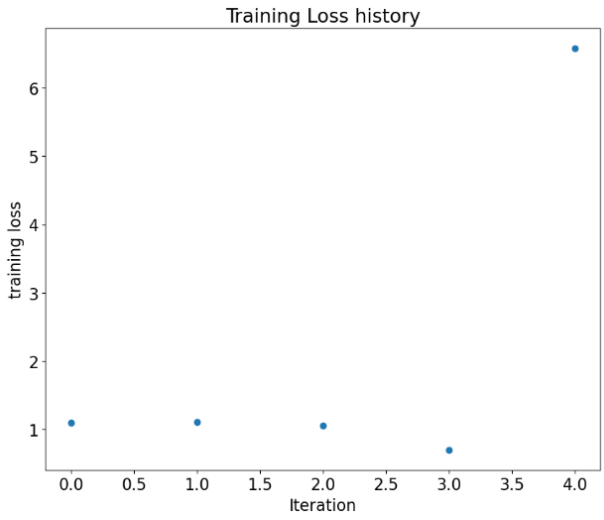
我們會發現 loss 並沒有在下降，這是因為 learning rate 太大導

致它在 loss function 上左右彈跳，如下圖所示



圖片來源：https://ithelp.ithome.com.tw/articles/10189086

在 learning rate = 1.8 的時候，只會導致左右彈跳，並不會有 loss 上升的問題，但如果再加大 learning rate，到 learning rate = 3，結果如下



可以看到 loss 有增加的情況。

- 結論：learning rate 的調整對於訓練模型來說非常重要，合適的 learning rate 可以讓訓練過程順利找到 global minimum，而太小或太大的 learning rate 會讓 loss 無法順利下降，甚至出現不降反升的現象。

五、 心得總結

這一次作業主要的內容是實作 forward propagation 和 back propagation，透過 computation graph 的幫助，讓一個原本很複雜的微分問題變得相對容易解決。

教授上課時有提到做 gradient descent 時會發生的問題，因此我也透過這次作業的機會來了解一下詳細內容。透過網路搜尋，我了解到梯度消失與梯度爆炸的原因與解決辦法。原本我以為過大的神經網路準確度部會太好的原因在於 overfitting，但是瞭解了梯度消失與梯度爆炸後才知道，原來過大的模型有可能造成 loss 無法順利降低的問題，才導致準確度不如小一點的神經網路。

最後透過調整一些 training 的參數，成功的讓 loss 再一次的降低。在這過程中，我實際的看到訓練次數與 learning rate 對訓練模型的影響，也了解到 local minimum 對於訓練模型時會帶來什麼樣的困擾。

六、 Reference

[1] WenWei Kang “Back-propagation” https://medium.com/ai-academy-taiwan/back-propagation-3946e8ed8c55

[2] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/

[3] ZZY_dl “深度學習筆記(五): 學習率過大過小對於網路訓練有甚麼影響以及如何解決” https://blog.csdn.net/m0_51004308/article/details/113449233