EAI

Lab 1
Report

系級	113 電機乙
學號	F64096114
姓名	郭家佑

- **如何實作各個 layer**

我總共設計了三層 layer，包含 input layer，一層 hidden layer，一層 output layer，由於 input 與 output 的 neuron 數被固定住了，所以我中間的 hidden layer 就取 784 和 10 的中間值附近，不要讓他突然縮減太快。

一開始我先初始化各層的 parameter，參考網路前輩的建議下，使用了高斯分布的隨機函數初始化，拿去給之後的 layer 使用。

其中 forward propagation 較為簡單，InnerProduct 造著 NN 架構給他做 MACs 動作就好了，只是 tensor 要稍微對一下，而我是採用 sigmoid 當作 activation function，想說 backward 會比較好微分，接著是 softmax，這裡我卡了半天，原因是我採用網路上一則較安全避免 overflow 或除 0 的公式:

y = np.exp(x - np.max(x))/ np.sum(np.exp(x))

但最後 debug 發現出來的值非常怪就改回原本的了。

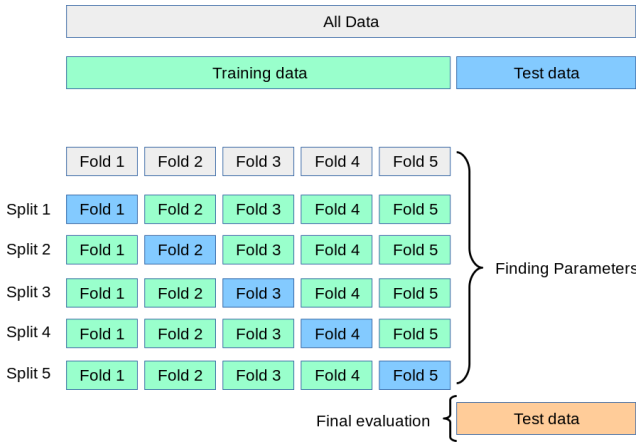
Loss function 我則是用 cross-entropy，這裡我也卡很久，原因是 1.參考錯 loss function 成只有 binary 結果的 2.ground truth 沒代 one hot encoding 而是 index value，修改完後就正常許多了。

而 backward 就難許多，原因是要一直對她的 tensor，而我

這個 python 新手是參考一篇教 np.dot()、np.multiply()、np.matmul()
https://blog.csdn.net/FrankieHello/article/details/103510118
才知道 numpy 有這些函示可以代。InnerProduct 和 softmax 就照著講義的公式把它刻出來，而 sigmoid 的則是將 dEdy 乘上 sigmoid 的微分(=自己*(1-自己))，就完成 back propagation。

而 update parameter 則是利用 gradient descent 來將 backward propagation 得到的 gradient 乘上自己設定的 learning rate 再與原先的 parameter 相減，拿去更新 parameter。

最終搭建自己的 NN，我 epoch 設定為 10，另外將 validation 的資料量設為 training 的 1/10，並利用 cross validation 將資料在 10 次 epoch 中能夠輪流都有用到，避免針對其中一小塊 validation data 去優化。底下就分為 training(54000 次)和 validation(6000 次)。Training 需要 forward+backward+updating 才算一輪，而 validation 只能拿來測試，所以只能跑 forward 去看結果(loss+accuracy)，最後跑完 60000 次再 testing。

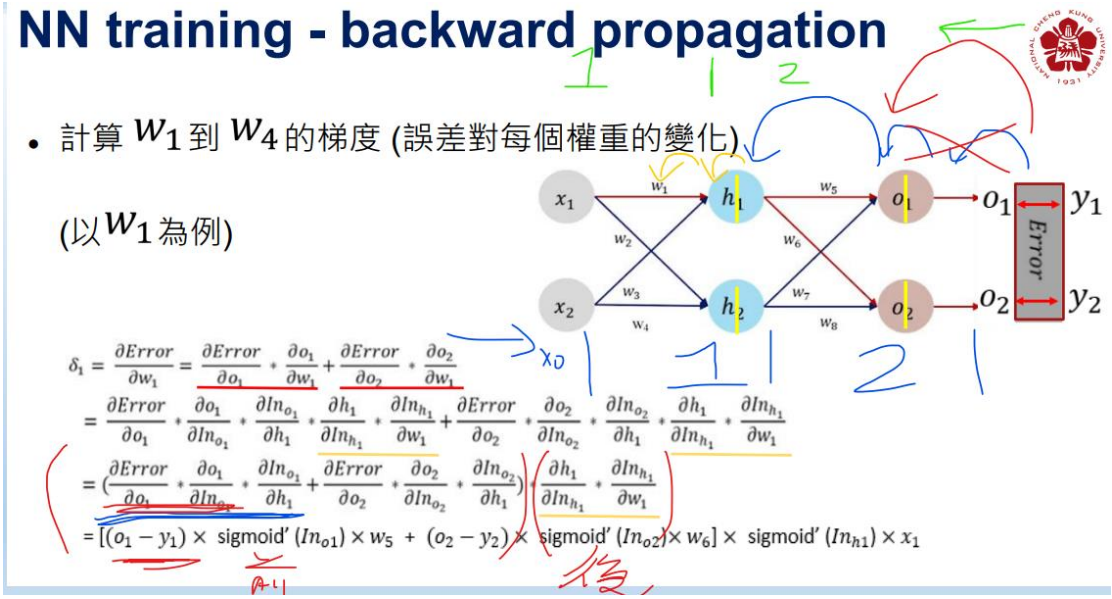


- **forward / backward 如何進行**

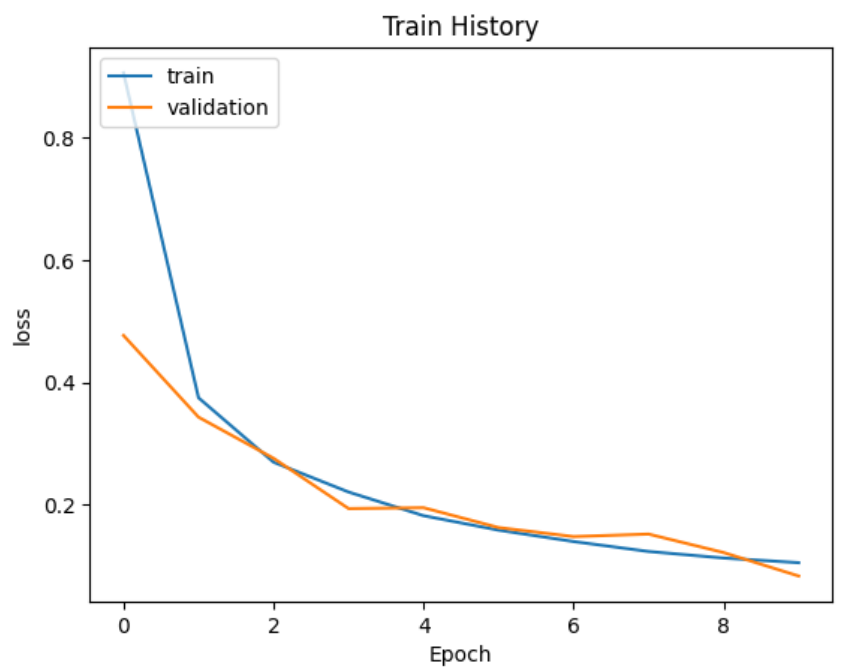
  forward 的重點是在傳遞到最後的結果 y，當然中間的各 neuron 的 output 也必須記起來給 backward 用。

  而 backward 的精華是在用 forward 得到的 Y 與 ground truth 去算 loss function，一層一層利用 chain rule 往回推每個參數對 loss 的梯度，並利用 gradient descent 往低點調整。

  可以參考助教精心製作的講義與我精美的筆記

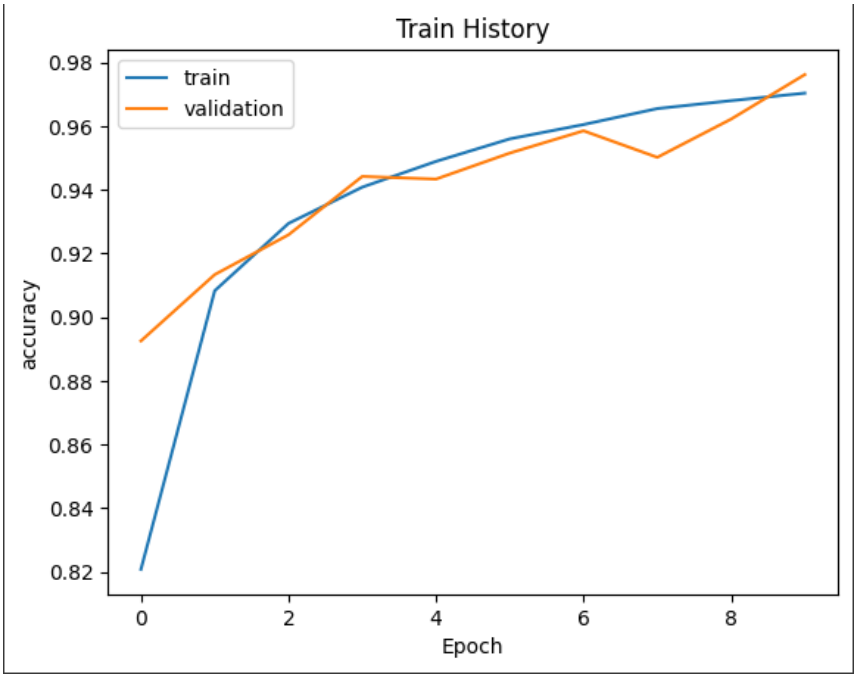


- **截圖並說明各項結果(包含 accuracy 和 loss 圖表(plot)的結果)**
  - Loss



可從圖表中看出 train_loss 從 epoch0 的 48000/54000 快速在下一個 epoch 降到 20000/54000，可知道這次用來 updating 的 gradient 對整體的 parameter 影響很大，後面則是隨著越快要到 optimal 而 loss 改善誠到降低。

  - accuracy plot



可由上面 loss 的結果反推 accuracy，兩者有著負相關的關係，epoch0 到 epoch1 的 accuracy 大幅上升是因為模型的 updating 在這個 epoch 非常給力的潮 optimal point 跨一大步，後續隨著 epoch 增加而越來越準。

- Training accuracy

```
epoch:  0 train_loss:  48934.75208252075 val_loss:  2861.447608209934 train_accuracy:  0.8207592592592593  val_accuracy:  0.8925
epoch:  1 train_loss:  20251.57845485265 val_loss:  2060.1704716892323 train_accuracy:  0.9082592592592592  val_accuracy:  0.9133333333333333
epoch:  2 train_loss:  14565.776612674385 val_loss:  1656.8715106186041 train_accuracy:  0.9294074074074075  val_accuracy:  0.9258333333333333
epoch:  3 train_loss:  11931.960905266322 val_loss:  1163.5126367559265 train_accuracy:  0.9407962962962962  val_accuracy:  0.9441666666666667
epoch:  4 train_loss:  9842.897804349886 val_loss:  1172.0189774999412 train_accuracy:  0.9488888888888889  val_accuracy:  0.9433333333333334
epoch:  5 train_loss:  8566.986722189482 val_loss:  977.267963536671 train_accuracy:  0.955962962962963  val_accuracy:  0.9515
epoch:  6 train_loss:  7561.881147468628 val_loss:  888.8446533311067 train_accuracy:  0.960462962962963  val_accuracy:  0.9585
epoch:  7 train_loss:  6684.8392052347035 val_loss:  913.3228200445398 train_accuracy:  0.9654814814814815  val_accuracy:  0.9501666666666667
epoch:  8 train_loss:  6105.835663287172 val_loss:  732.2169385726182 train_accuracy:  0.9679629629629629  val_accuracy:  0.9621666666666666
epoch:  9 train_loss:  5696.153603164086 val_loss:  502.02267253986753 train_accuracy:  0.9703333333333334  val_accuracy:  0.9761666666666666
```

- testing accuracy

```
test accuracy: 0.9439
```

由 testing 結果可看出模型沒有 ovevrfitting

- **遇到的困難及你後來是如何解決的(optional)**

我發現網路上查詢的資料很多都有錯，都是我試到最後才發現用最初的作法才最可行，而我解決的辦法就是把每一個不確定對不對的結果 print 他的 shape 和 value 來驗證是否與預期的一樣，加速 debug。

這個 lab 是我第一次親手寫 python，一開始連 list numpy 都不熟，所以花了好一大段時間查詢相關資料，而對 NN 的運作也是聽完助教講解才知道的，過程中一直遇到打擊，尤其是看到不管怎麼調 accuracy 都還是比我亂猜還低時，想把我擊退去按退選，好險最終有撐過來，但我要趕快去寫 lab2 和 paper review 了，deadline 一個接著一個。最後也感謝助教百忙之中抽空回答同學 email 的問題。