1-1

- *Simulate a Function*:
  - model1 架構 :

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 5)	10
dense_2 (Dense)	(None, 10)	60
dense_3 (Dense)	(None, 10)	110
dense_4 (Dense)	(None, 10)	110
dense_5 (Dense)	(None, 10)	110
dense_6 (Dense)	(None, 5)	55
dense_7 (Dense)	(None, 1)	6

Total params: 461
Trainable params: 461
Non-trainable params: 0

  - model2 架構 :

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 7)	14
dense_2 (Dense)	(None, 8)	64
dense_3 (Dense)	(None, 19)	171
dense_4 (Dense)	(None, 10)	200
dense_5 (Dense)	(None, 1)	11

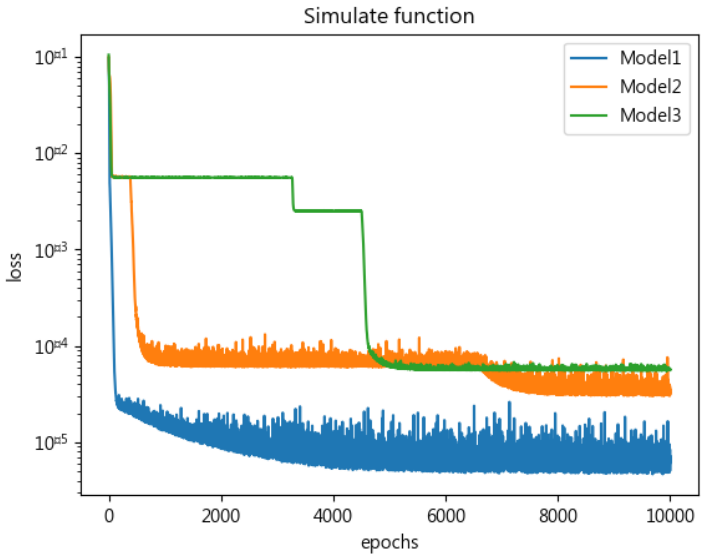
Total params: 460
Trainable params: 460
Non-trainable params: 0

  - model3 架構 :

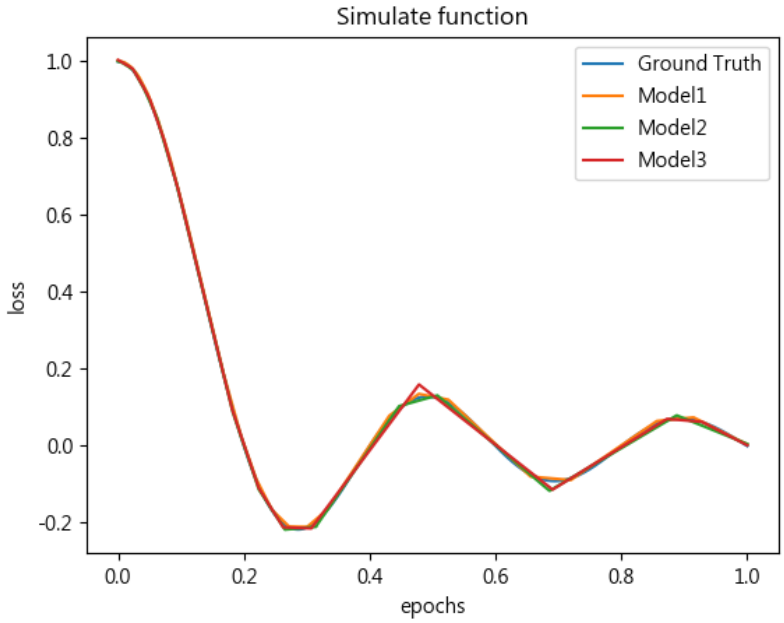
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 20)	40
dense_2 (Dense)	(None, 19)	399
dense_3 (Dense)	(None, 1)	20

Total params: 459
Trainable params: 459
Non-trainable params: 0

  - 函數 : $\frac{sin5\pi x}{5\pi x}$
  - training loss:



- predicted function curve and the ground-truth function curve.



- Comment on your results. (1%)
  - 在loss部分明顯看出層數越深的loss降的越早，而最後也降到越低。在跟原本函數值比較的部分，雖然最寬的model因為訓練的epoch蠻多的關係，導致於他把整個資料硬學了起來，但是我們還是可以仔細地看到他在原本函數應該是圓滑曲線的部分有著稜角，而其他的model的稜角則是沒有突出的這麼誇張。
- Bonus:
  - 函數：$y = cos(x) * x,\ x = [0,\ 5]$

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 5)	10
dense_2 (Dense)	(None, 10)	60
dense_3 (Dense)	(None, 10)	110
dense_4 (Dense)	(None, 10)	110
dense_5 (Dense)	(None, 10)	110
dense_6 (Dense)	(None, 5)	55
dense_7 (Dense)	(None, 1)	6

Total params: 461
Trainable params: 461
Non-trainable params: 0

  - model1 架構：
  - model2 架構：

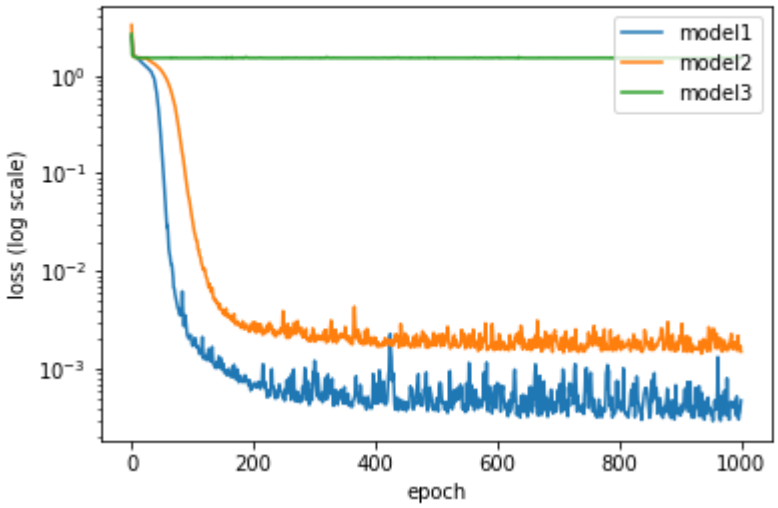
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 10)	20
dense_2 (Dense)	(None, 15)	165
dense_3 (Dense)	(None, 12)	192
dense_4 (Dense)	(None, 6)	78
dense_5 (Dense)	(None, 1)	7

Total params: 462
Trainable params: 462
Non-trainable params: 0

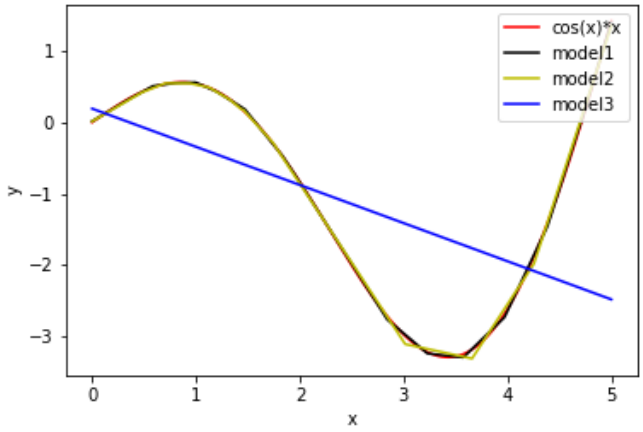
- model3 架構：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 153)               306
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 154
=================================================================
Total params: 460
Trainable params: 460
Non-trainable params: 0
```

- training loss 圖：



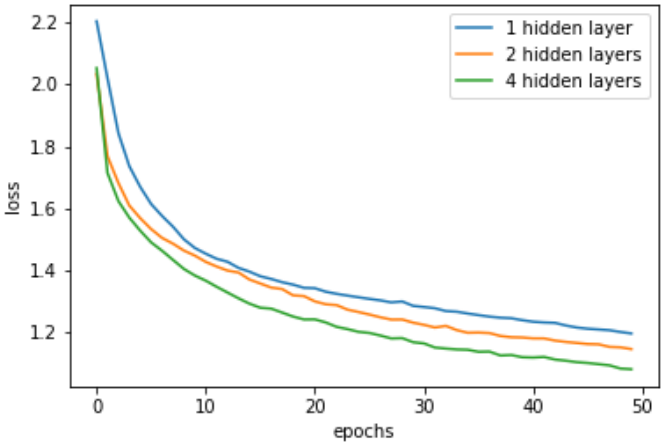
- 函數圖形和各 model 預測的圖形：

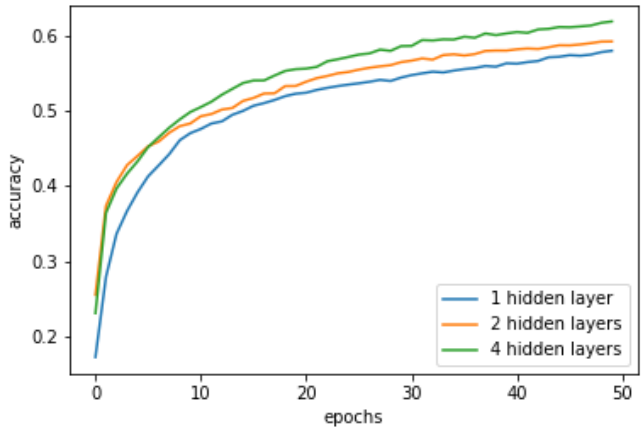


- Comment: 在我的實驗中可以很明顯地看出，幾乎相同參數的形況下，較深的模型確實有較好模擬函數的能力，所學出來的曲線也較為平滑。

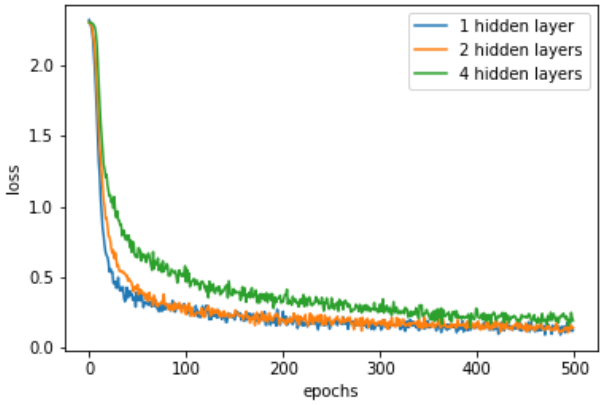
- *Train on Actual Tasks*:
  - Task: CIFAR10
  - Models
    - 架構：input -> conv2d -> max pooling -> hidden layers -> output

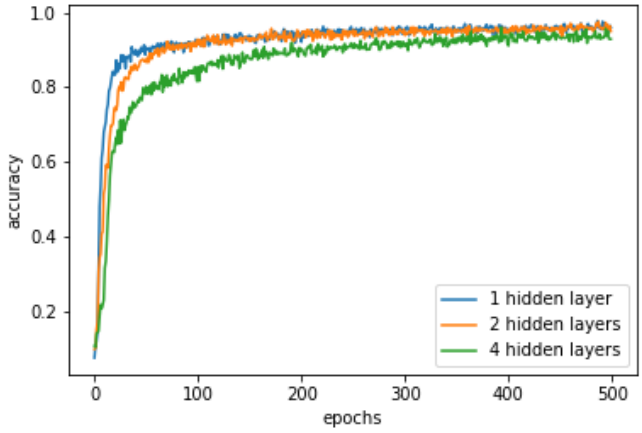
- model1: 1 hidden layer, 總參數為 28792
- model2: 2 hidden layer, 總參數為 28027
- model3: 4 hidden layer, 總參數為 28616





- Comment on your results. (1%)
  - 參數量相近的情況下，比較深的 model 可以有比較好的表現。
- Another task: mnist
- Model: input -> conv2d -> Dense (不同數量的 hidden layer) -> output
  - model1: 1 hidden layer, 總參數為 3197
  - model2: 2 hidden layer, 總參數為 3175
  - model3: 4 hidden layer, 總參數為 31

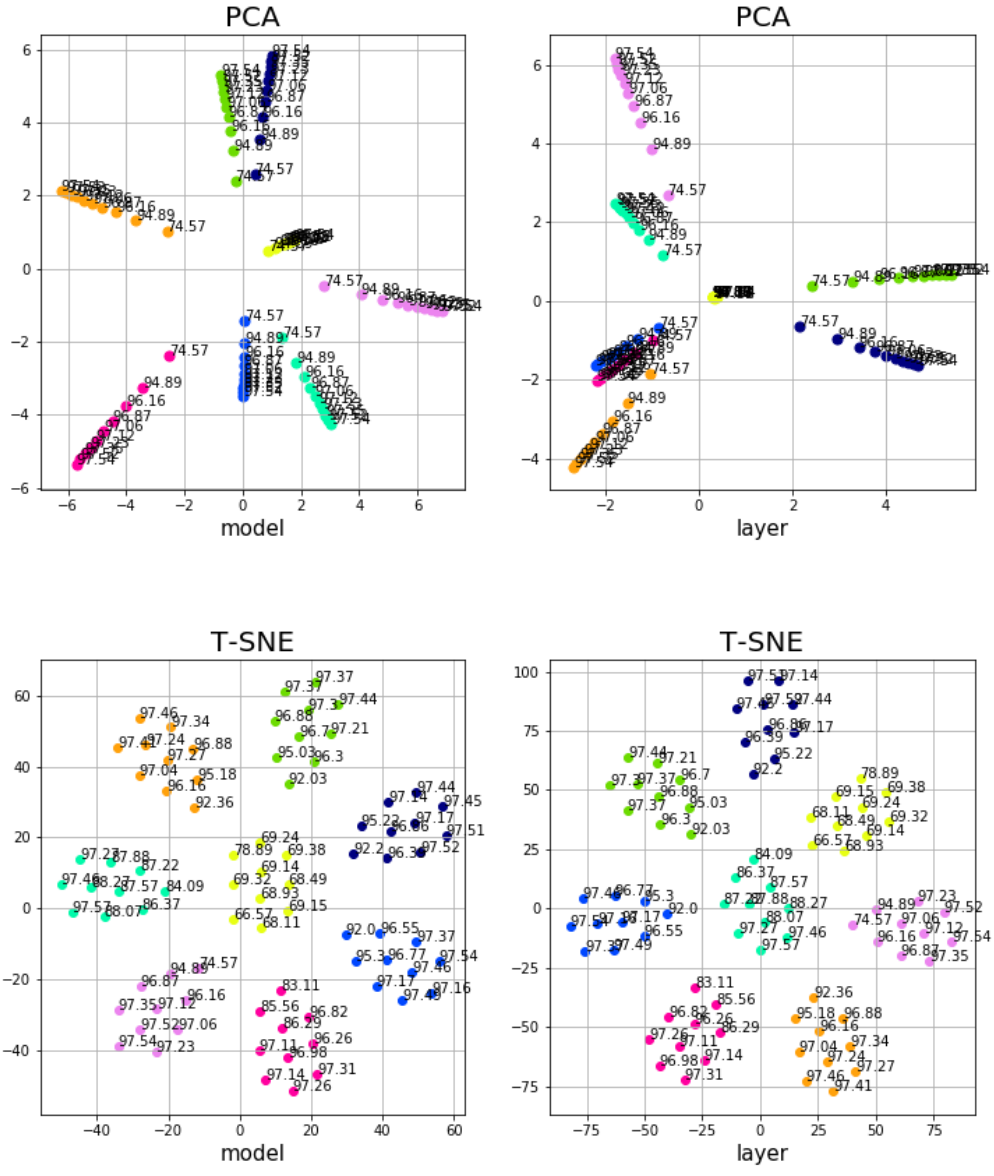




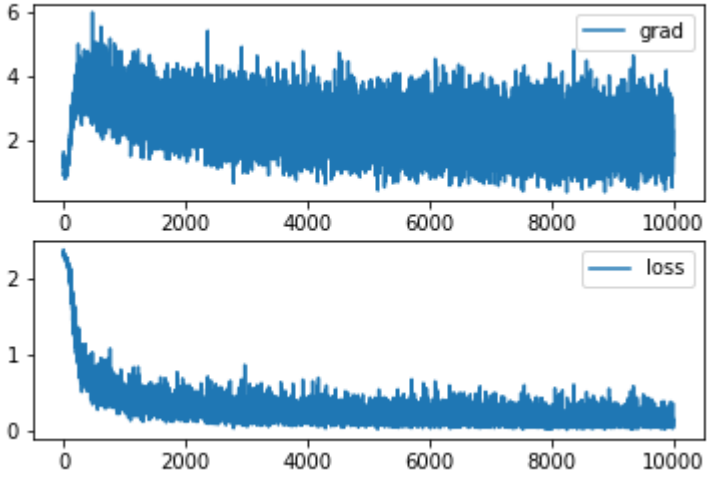
- Comment:
  - mnist 上的結果就比較不如預期，一層和兩層 hidden layer 的結果差不多，四層 hidden layer 的結果還更差。我的推測是因為 mnist 太好訓練，很簡單的 model 就能有很好的結果，因此更複雜的 model 能進步的空間相對小很多，導致有時反而還變差。

1-2

- *Visualize the optimization process*.
  - 設定:
    1. 每 3 epoch 紀錄 weights
    2. optimizer: SGD
    3. dimension reduction 使用: PCA, T-SNE

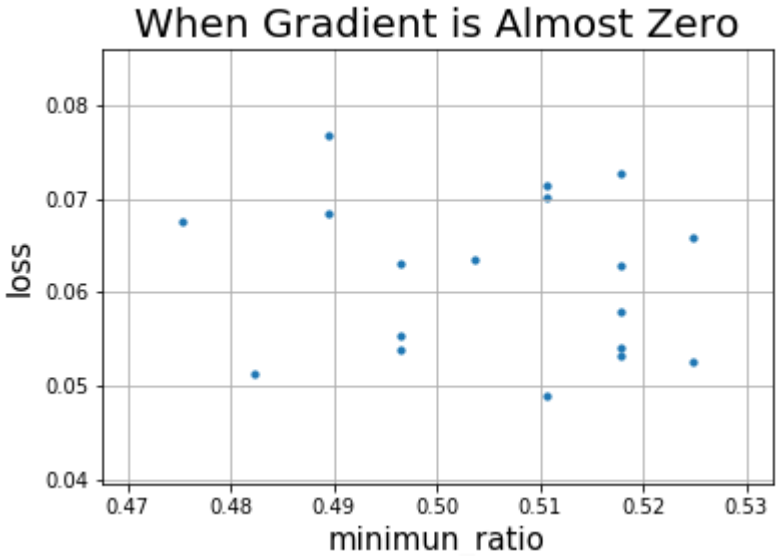


  - 從圖上來看可以發現每次optimize的參數都不盡相同，但都可以有不錯的準確率(97%↑)，可以猜測可能有很多個local minimun
- *Observe gradient norm during training.*
  - gradient norm 對 iterations 和 loss 對 iterations 的圖：



- Comment: 當gradient norm越大，loss震盪幅度也變大，training的過程似乎沒有遇到saddle point

- *What happens when gradient is almost zero?*
  - 訓練方法：
    - 首先，用一般的方式訓練
    - 再將原本的目標函數換成 gradient norm，繼續訓練，直到 gradient norm 到零。

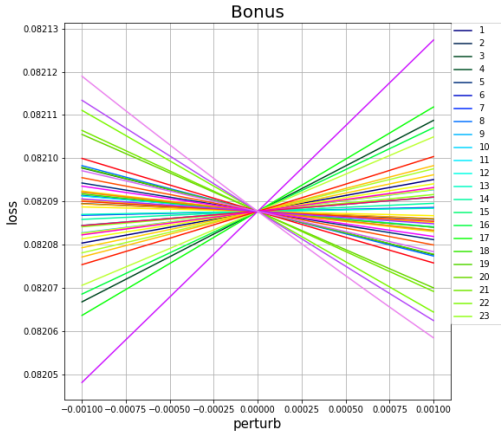


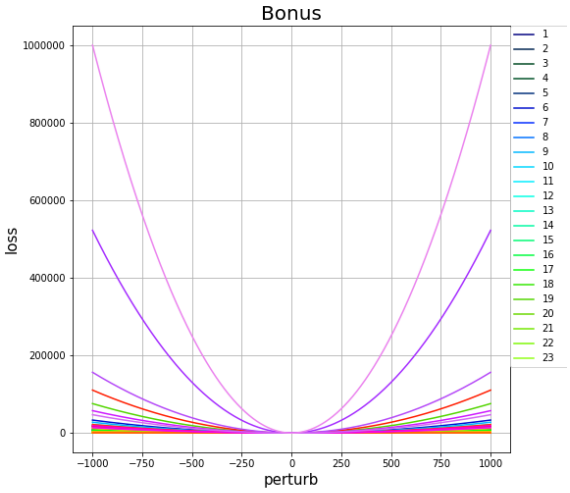
  - Comment your result. (1%)

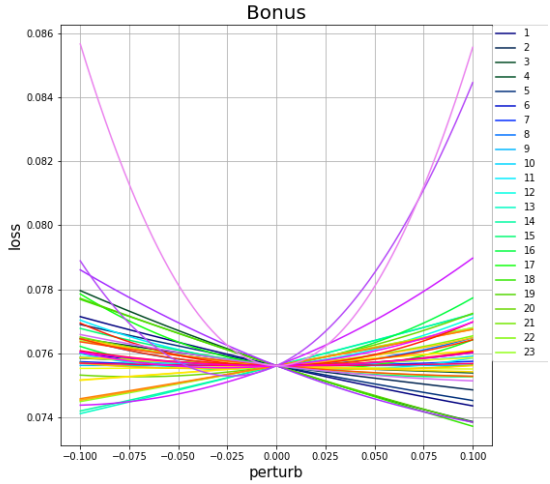
    從結果來看loss和miniratio的關係沒有很明顯，在訓練的過程中loss和

gradient　norm似乎沒辦法在繼續下降，如果能夠更逼近於零或許關係會更清楚，這張圖是做simulate　function　$\frac{\sin 5\pi x}{5\pi x}$　在上面，並計算Hessian matrix的eiganvalue，也有可能找到的點都是local minimun不是saddle point只是loss沒有非常低

- Bonus
  - Concretely describe your method and comment your result.
    - 對每個parameters加上一些noise之後對loss做觀察，發現在極小的擾動下，loss呈近似線性變化





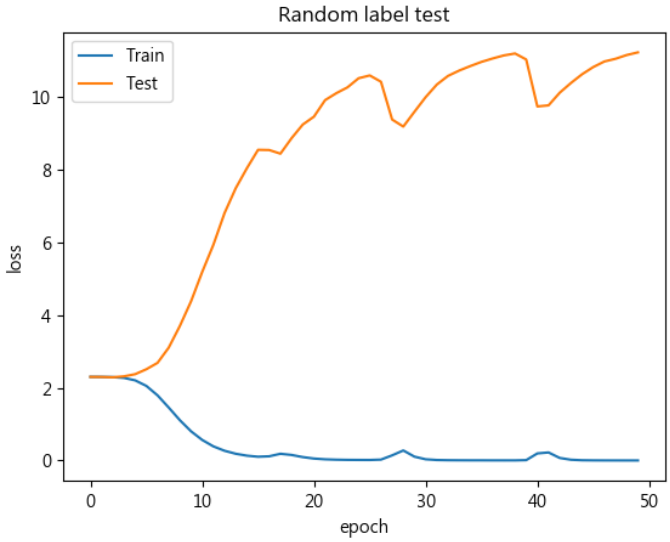


<u>1-3</u>

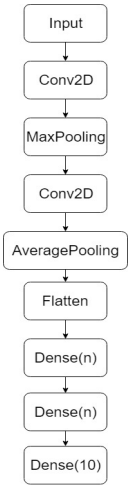
- *Can network fit random variables?*
  - <u>設定</u>：
    - Target : MNIST
      model : Conv2D->Flatten->Dense, and softmax in last layer

```
model = Sequential()
model.add(Conv2D(32,3,3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(64,3,3, activation='relu'))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

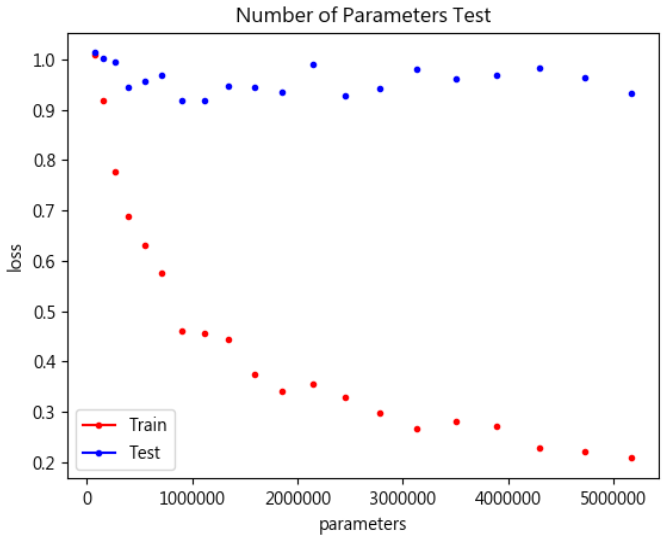
- Plot the figure of the relationship between training and testing, loss and epochs. (1%) loss為 categorical_crossentropy

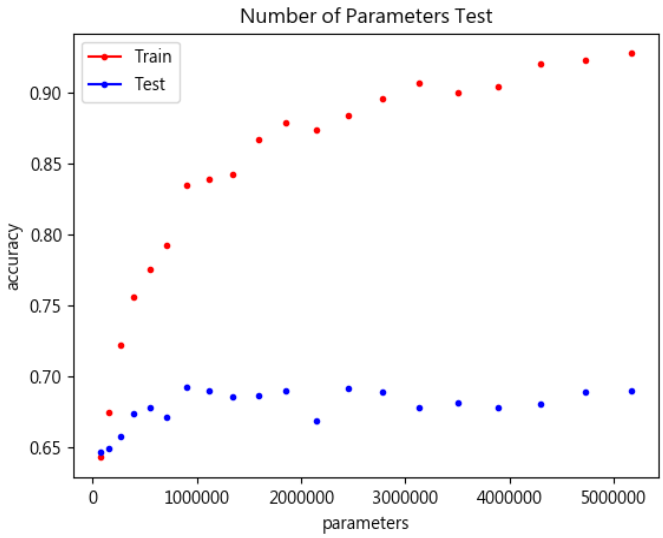


- *Number of parameters v.s. Generalization*
  - Setting:
    - Target : cifar10, model 如下，n = 100, 200....2000, 總共 20 models



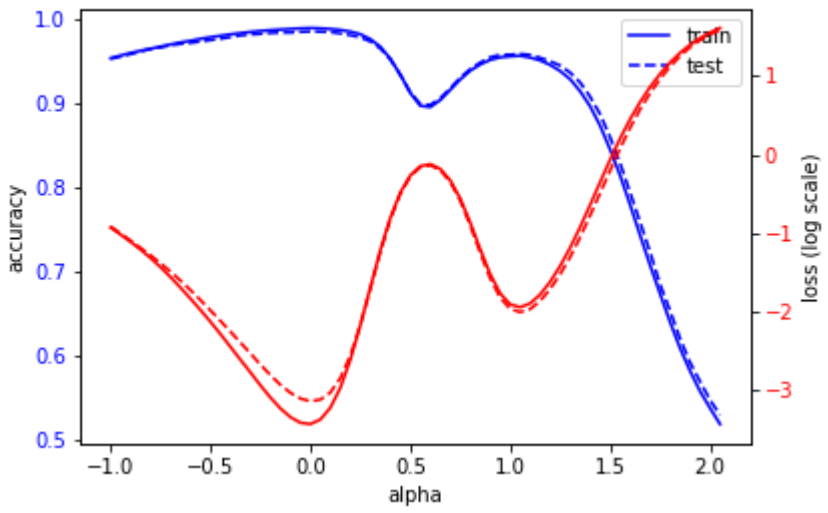
  - Plot the figures of both training and testing, loss and accuracy to the number of parameters. (1%)



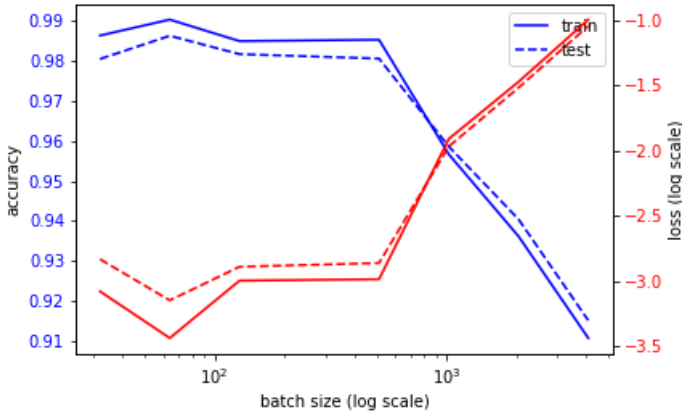


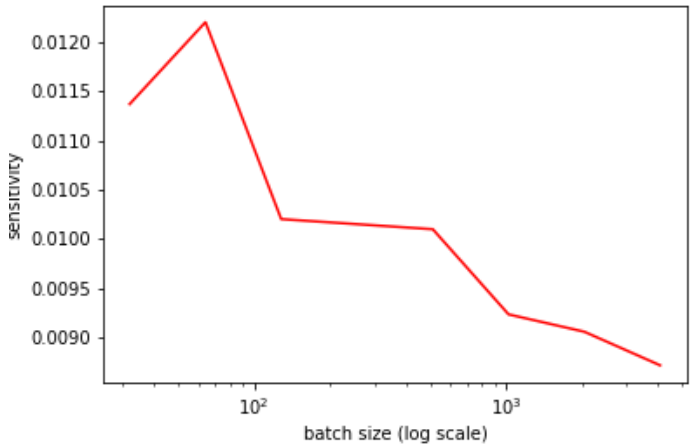
- Comment your result. (1%)
  - 在training過程中可以發現當參數量越多時，training set跟testing set的loss及accuarcy差距越大，代表著可訓練的參數越多時，很容易的就會overfit training set，導致training loss不斷的下降時，testing loss不但沒有下降還反升。

- *Flatness v.s. Generalization*
  - Part 1:
    - task: mnist
    - learning rate: 1e-4, optimizer: adam
    - model 架構:
      input-> conv2d -> max pooling -> dense -> dense (output)
    - training approach:
      model1: batch size = 64
      model2: batch size = 1024
    - training and testing, loss and accuracy v.s. interpolation ratio



        - alpha = 0 的地方是 model1，而 alpha = 1 的地方是 model2。兩個 model 分別找到了不同的 local minima，前者找到的 local minima 的附近有稍微比較平一點，但是差別不大。圖中也可以觀察出 training set 和 testing set 的 local minima 是在不同的地方，model1 較容易看出來，但也是沒有很明顯。
    - Part 2 :
        - task: mnist
        - learning rate: 1e-4, optimizer: adam
        - model 架構: input-> conv2d -> max pooling -> dense ->output
        - training approach:
          使用以下不同的batch size訓練: 32,64,128,512,1024,2048,4096





        - batch size 會影響到 accuray，我的實驗中當 batch size = 64 時，會有最好的結果，但是它卻有最大的 sensitivity。而由兩圖比較可以得知，小的 batch size 卻有較大的 sensitivity，可以推斷我的結果中 sensitivity 和 sharpness 並沒有太大關係。此結果也不是太意外，因為 sensitivity 僅只是一個猜測 sharpness 的方法，並無明確的關係。

- 分工表：
    - 蔡昀達：1-2
    - 謝宏祺：1-1-1, 1-3-1, 1-3-2
    - 朱柏澄：1-1-1, 1-1-2, 1-3-3