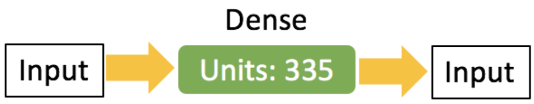
MLDS HW1 Report

許芯瑜、李祐賢、熊展軒

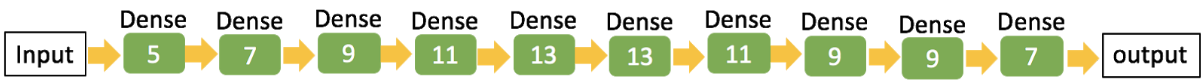
HW1-1:

- **Simulate Functions**
    - Model: (Every dense layer with activation function 'relu')
        - model_1 (1006 parameters with 1 hidden layer):

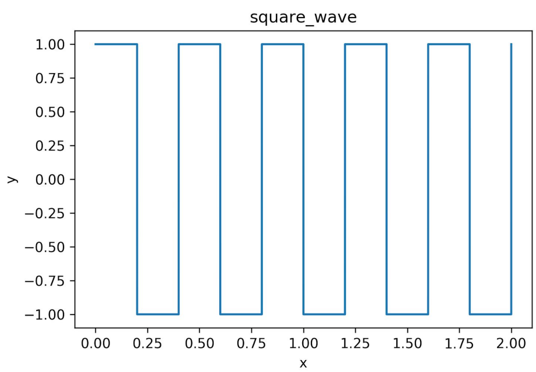


        - model_4 (1004 parameters with 4 hidden layer):

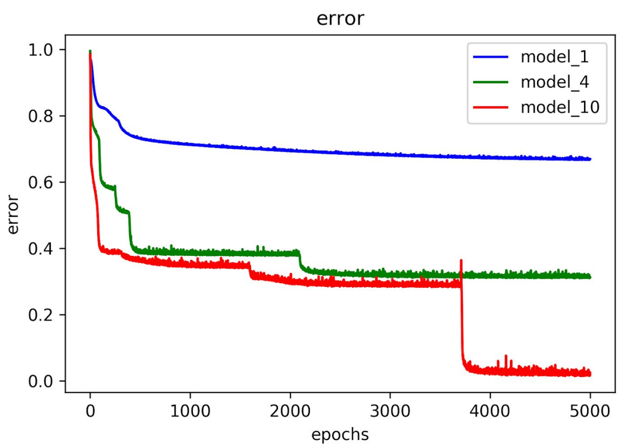
        - model_10 (1002 parameters with 10 hidden layer):



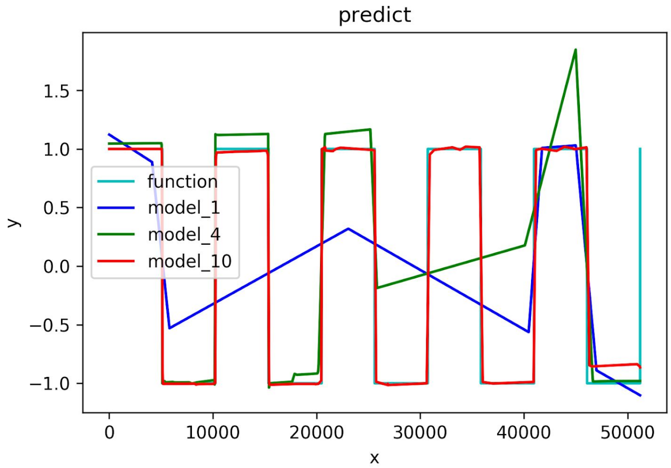
    - The range of all functions is between 0 and 2
    - Function 1: Square wave
        - Function:



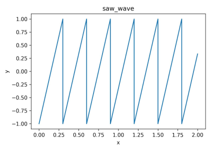
        - Training loss:



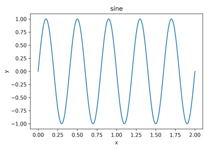
- ■ Predictions:



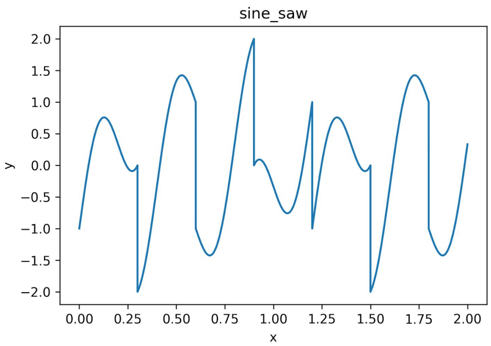
- ○ Function 2: Saw wave + Sine function
  - ■ Function:



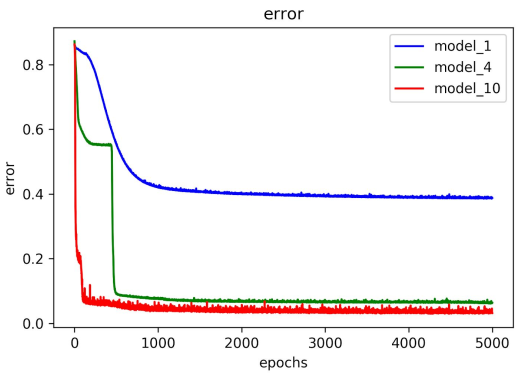
+

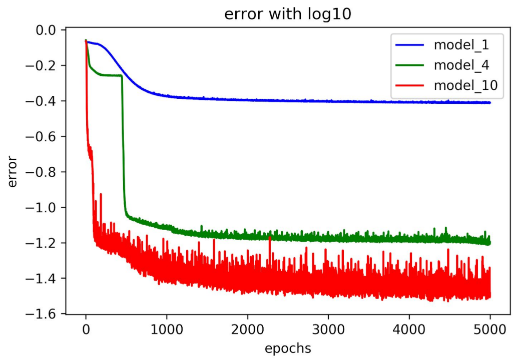


=

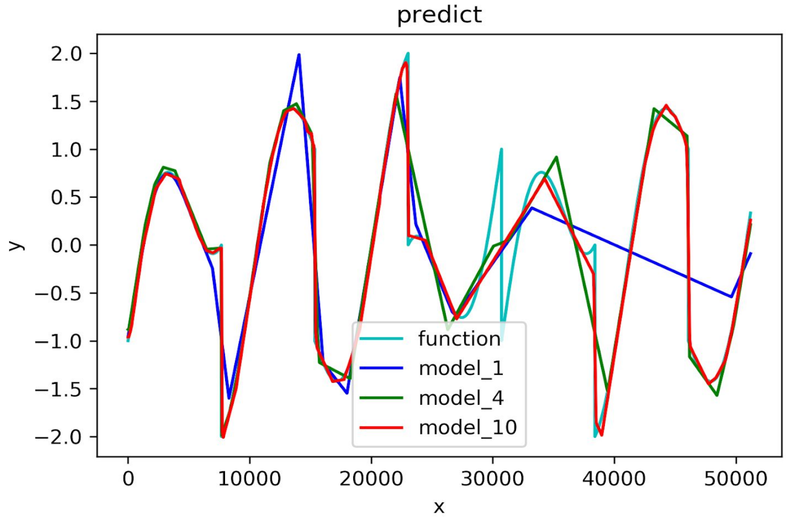


- ■ Training loss:

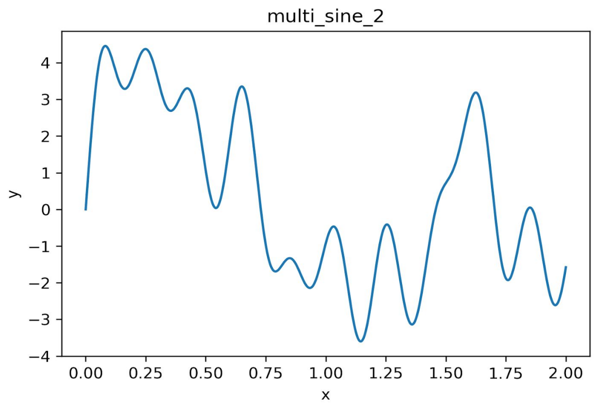




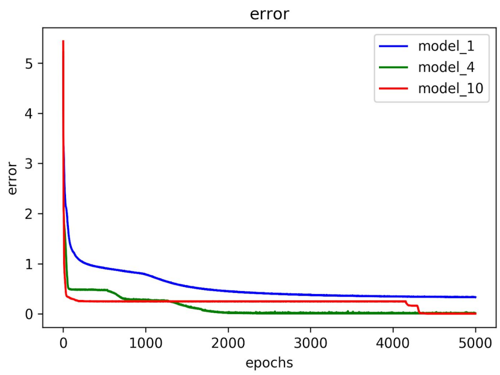
- Predictions:

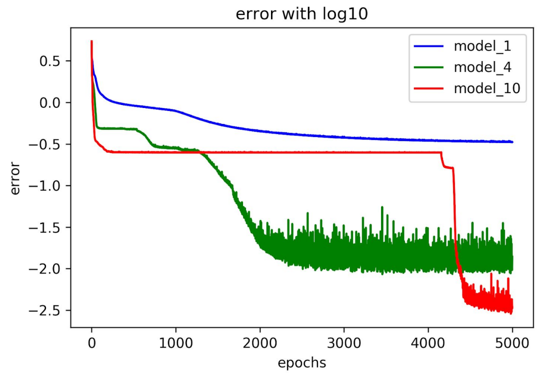


- Function 1: Multiple sine function
  - Function:
    sine1(x) = sine(x) with wavelength 1, height 1
    y = sine0.2 + sine0.3 + sine0.5 + sine0.7 + sine1.1 + sine1.3 + sine1.7 + sine1.9

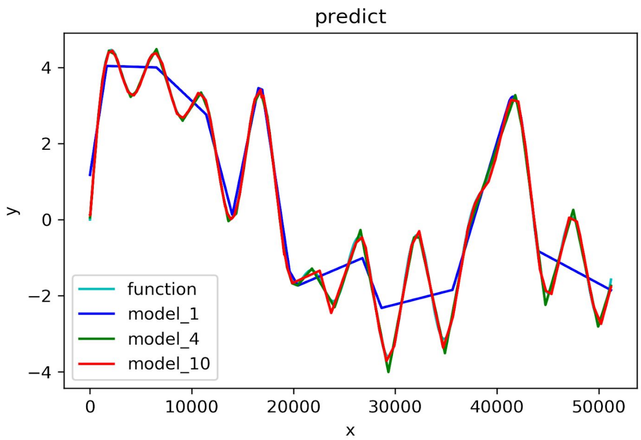


  - Training loss:





■ Predictions:

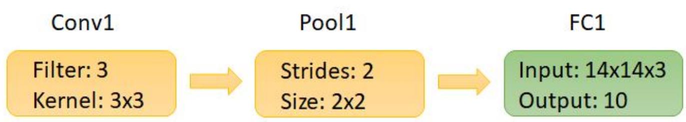


○ Comment:
在所有的case當中越深的model的效果都是越好的，通常也較快收斂，但在Function 3中在epoch數不夠多的情況下，較深的model反而會有更大的loss，且loss常常會有卡住後驟降的現象，應該是陷入gradient很小的地方(not sure)。

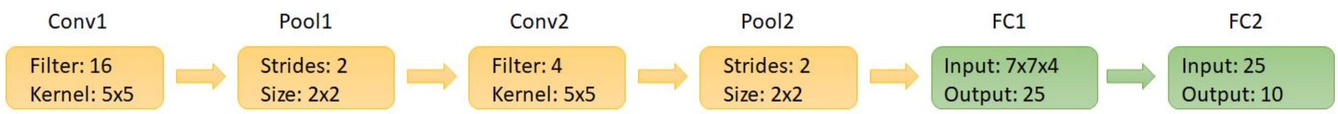
● Train on Actual Tasks

○ Experiment settings:

■ Shallow model(參數量: 5910):

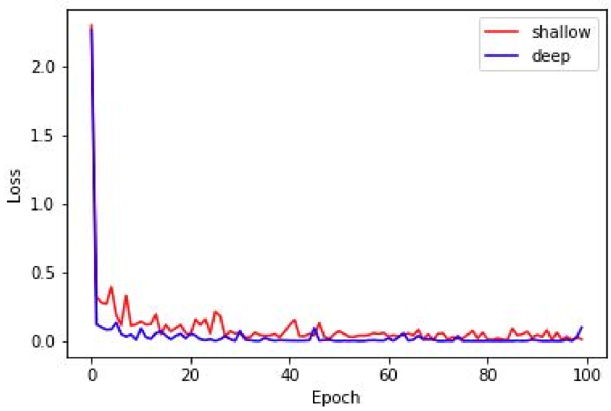


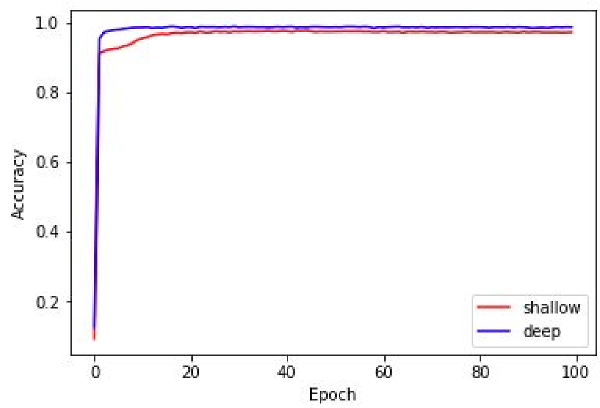
■ Deep model(參數量: 5670):



■ Task: MNIST

○ Visualize result:



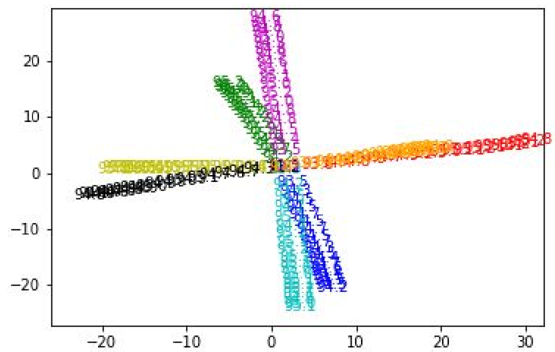


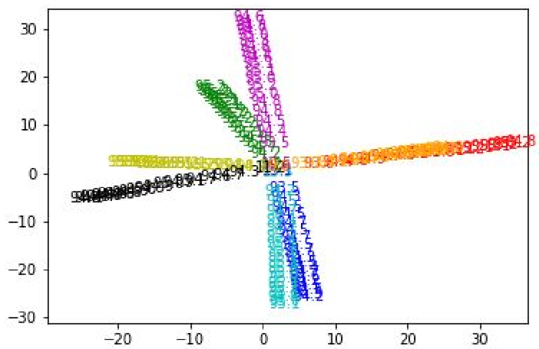
○ Comment:

從Loss的圖可以看出深的model收斂淺的model快，且最終收斂的值比淺的model更低。由視覺化的結果可以發現深的model可以更快找到local minimum，且找到的minimum是比淺的model找到的local minimum更好的。
從Accuracy的圖可以看出深的model比淺的model更快收斂，且最終得到的accuracy也比淺的model來的更好。
由此兩張圖可以了解到，即使是在真實的task且同樣參數量的情況下，深的model可以表現得比淺的model更好。

HW1-2:

- **Visualize the optimization process**
    - Experiment settings:
        - Cycle: 5 epochs one cycle
        - Optimizer: Adam
        - Dimension reduction method: PCA
    - Visualize results (Left: layer 1 & Right: whole model):

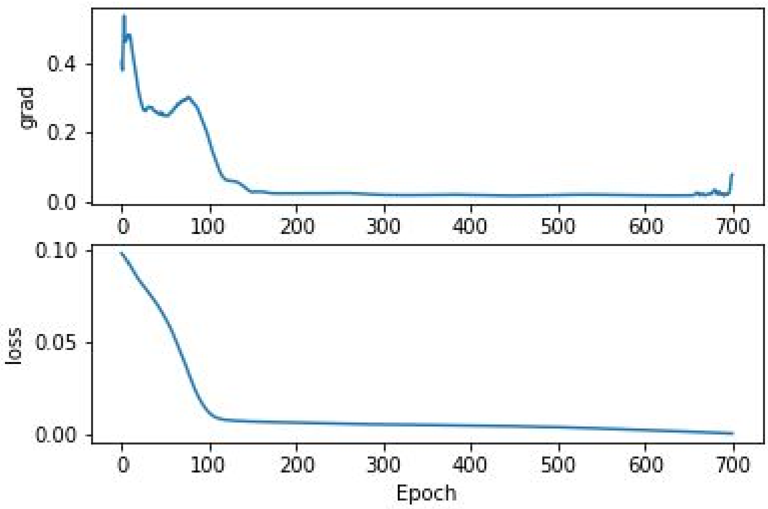


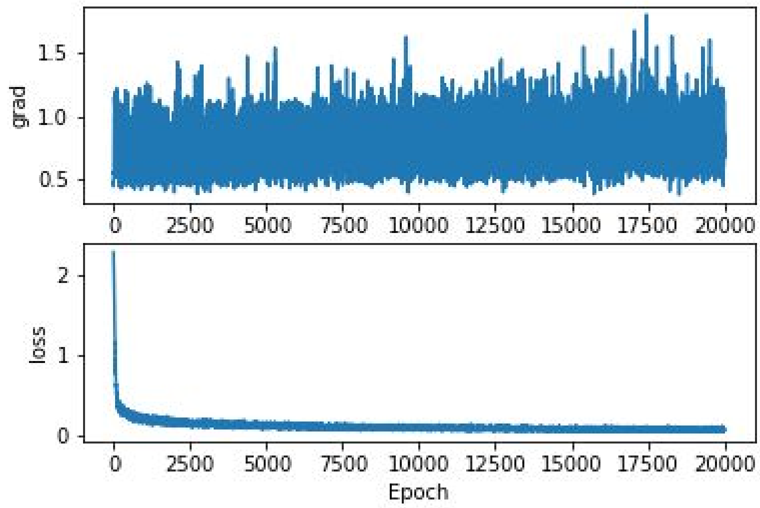


    - Comment:
      由上兩張圖可以看出每個training event的optimizer過程都是越來越往外移動的，這代表他們找到的solution都是在外圍的地方。另外，可以看出每個training event因為初始值不同所以optimizer的方向也不同，找到的solution當然也不會相同，這也可以讓我們了解在deep learning中是有許多local minimum的。而且在這些local minimum我們得到的accuracy其實是逼近100%的，所以我們也可以了解local minimum可能和global minimum差距並不大，我們找到的local minimum其實已經夠用。

      在分別分析兩張圖，可以發現只看layer 1的optimize方向和看whole model的optimize方向其實差距並不大，我們認為應該是layer 1的參數在整個model中的重要程度比較大。
- **Observe gradient norm during training**

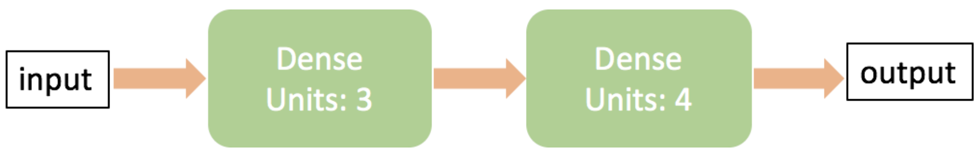
- Visualize results:





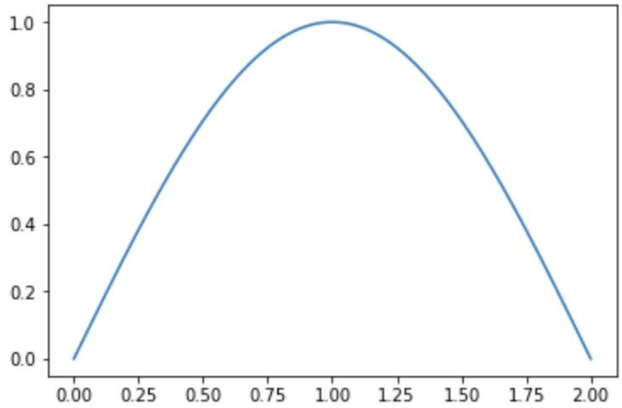
- Comment:
  在mnist中(上圖),gradient隨著loss下降而降低,而在function中(下圖), gradient部隨著loss下降而降低.

- **What happens when gradient is almost zero**
  - Model:



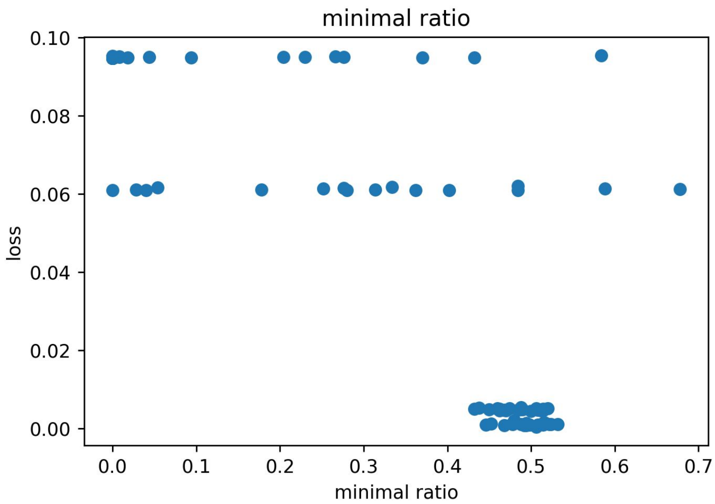
  - Experiment settings:

- Train on sine function (half sine wave in [0, 2]):



- Adam optimizer.
- Train 150 epoch, but if gradient norm is not lower than 0.03, continue training until 750 epoch.
- Use sampling method to get minimal ratio, and we sample 500 point.
- Train 100 run.

○ Figure:



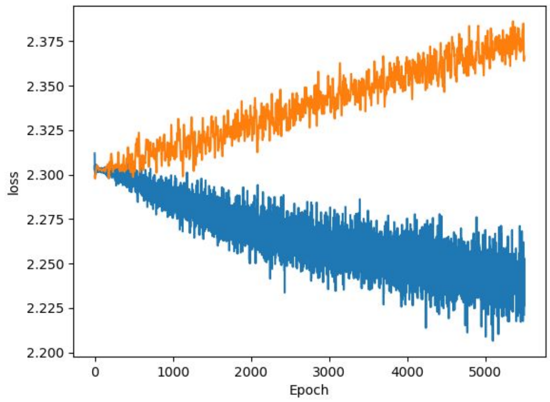
○ Comment:

在loss 很大（約0.95）的時候，minimal ratio 通常是非常小的，在左上角minimal ratio 為 0 的地方其實聚集了大量的點（minimal ratio < 0.01 的點有29個，而loss > 0.08的只有38個）。但是在loss 偏大時sample 出來的minimal ratio 很不穩定，可能是sample 不夠多或是該點處於一個高原上，周圍的點大多跟他差不多的loss ，造成取樣後不穩定的現象。

從這張圖推測本案例loss function 經常存在loss 為0.06 及0.95的高原地形或saddle point 。

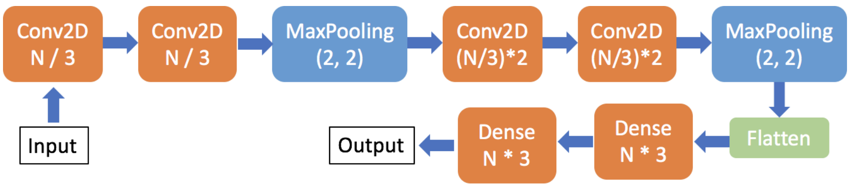
HW1-3:

- **Fit random variables**
  - Experiment setting:
    - Task: MNIST
    - Learning rate: 0.001
    - Optimizer: Adam
  - Visualize result:

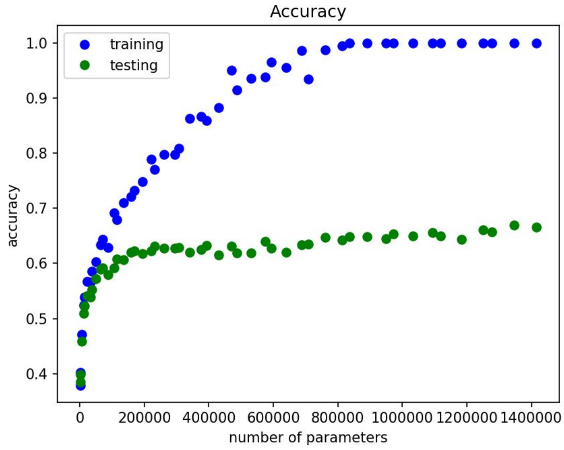


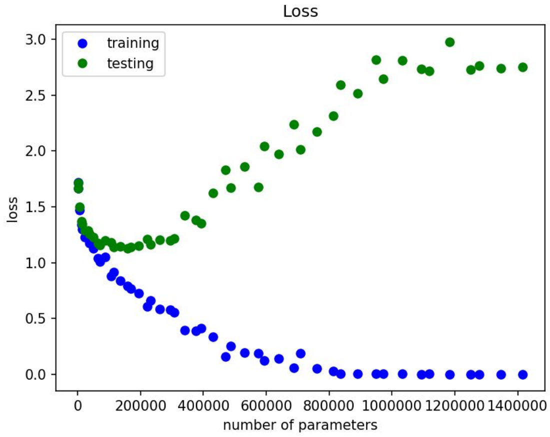
  - Comment:
    值得注意的是,我已經把每個label都打亂,但是training data還是降得下來,代表在面對一堆實際上無用的data時, 深度網路還是能夠去fit 它.

- **Number of parameters v.s. Generalization**
  - Experiment setting:
    - Task: CIFAR-10
    - 50000 trining data, 10000 testing data
    - Train 49 models
    - Train every model 50 epochs, with batch size 1000
  - Model:
    N : level of parameters in [3, 5, 7, …, 99]
    Different N represents models with different number of parameters



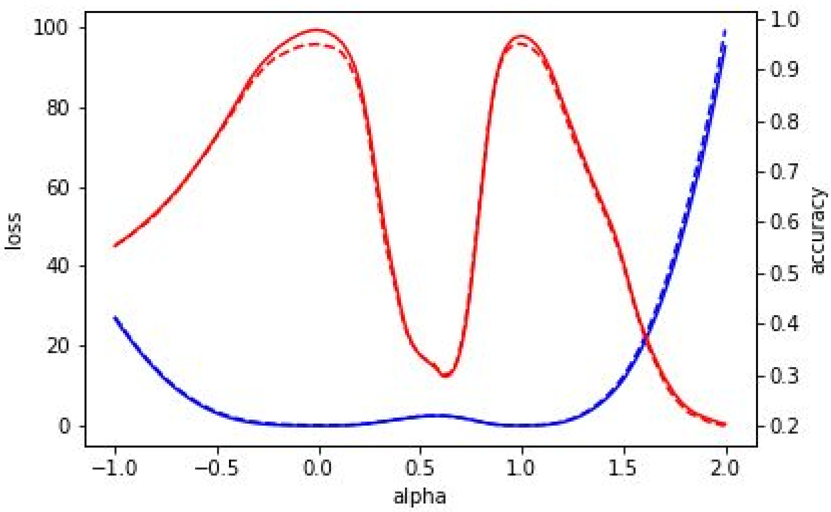
  - Parameters vs Loss/Accuracy:





- Comment:
  從Accuracy 的變化來看較多的參數並不影響Generalization 的程度，但Testing loss 在參數量多的時候卻有明顯的上升，看起來還是有明顯Overfitting 的現象，但是可能因為還沒有Train 到非常Overfit 最後結果經過Softmax 仍然沒有太大的變化，才會導致Accuracy 沒有下降。感覺在Train DNN 還是應該要稍微注意Overfitting 的問題。

- **Flatness v.s. Generalization (part 1)**
  - Experiment setting:
    - Task: MNIST
    - Training apporach: batch size 100 and 1000
  - Visualize result:

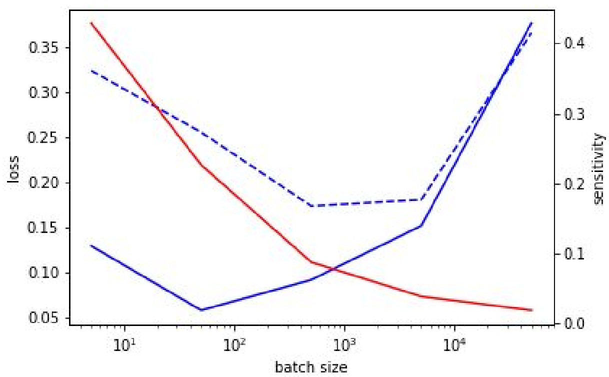


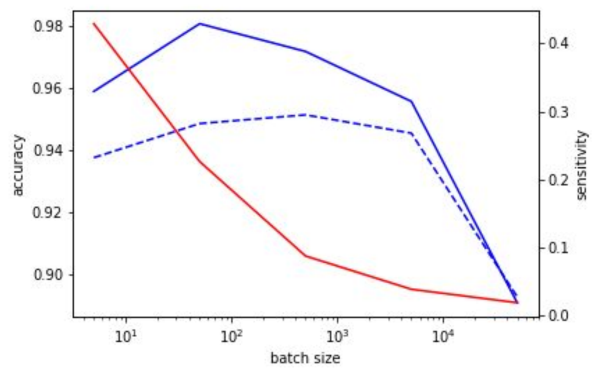
  - Comment:
    從上圖可以看出loss最低及accuracy最高的兩個地方分別是alpha為0及為1的地方，這兩個地方代表的是只用model 1的參數及只用model 2的參

數的結果。這與我們直觀的想法一致，畢竟我們已經將這個training approach訓練到最好的情況，冒然的調整參數一定會使performance下降。

由圖中也可以發現當alpha=0.5~2時的gradient會比alpha=-1~0.5更大，我們認為是因為model 2的batch size較大，所以會比較sensitive。在參數調整幅度相同的情況下，比較sensitive的model造成的影像會更劇烈。

- **Flatness v.s. Generalization (part 2)**
  - Experiment setting:
    - Task: MNIST
    - Training aproach: batch size 5、50、500、5000、50000
    - Sensitivity define: Frobenius norm of gradients of loss to input
  - Visualize result:





  - Comment:
    我們做出來的實驗與sensitivity理論結果背道而馳，照理來說batch size越大sensitivity應該要越大，但我們卻得到相反的結果。我們認為應該是我們採用的sensitivity的定義不是算output對input的微分，而是gradient對input的微分。

    這次的實驗我們採用了五個不同的training approach，可以發現最好的batch size不是最小也不是最大，而是一個比較中間的數(500)。從這個實驗我們可以發現要將model train得好，不管在深度、filter大小、learning rate，甚至是batch size都要精心調整才能讓深度網路發揮最大效能。

分工表:

- HW1-1: 1. 熊展軒 2. 許芯瑜
- HW1-2: 1. 許芯瑜 2. 李祐賢 3. 熊展軒
- HW1-3: 1. 李祐賢 2. 熊展軒 3. 許芯瑜