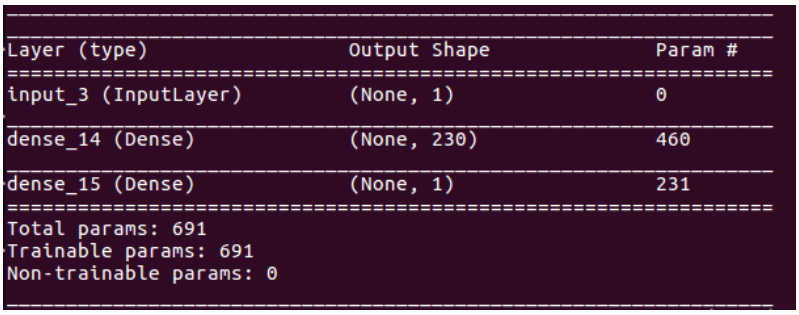
MLDS HW1 Report

姓名：羅章碩、鄒適文

- **1-1-1: Simulate a Function:**
  - **Describe the models you use, including the number of parameters (at least two models) and the function you use. (0.5%)**

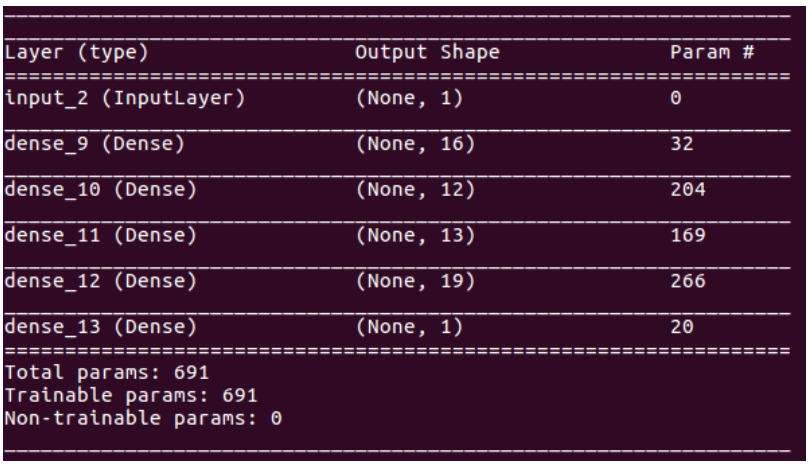
    我使用的是 DNN 的 network，總共使用了三種深度的模型，從深到淺分別(不算 input, output layer)有 7, 4, 1 層的模型，而他們總共的參數量都為 691，每層都是使用 relu 的 activation function，每層也都有使用 bias，使用的 loss function 為 mean square error，optimizer 是用 Adam，訓練資料為 0.01~1 之間共 sample 五萬筆資料，batch size 為 1024、epochs 為 15000。以下是三種 model 的架構圖：



Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 1)	0
dense_14 (Dense)	(None, 230)	460
dense_15 (Dense)	(None, 1)	231

Total params: 691
Trainable params: 691
Non-trainable params: 0

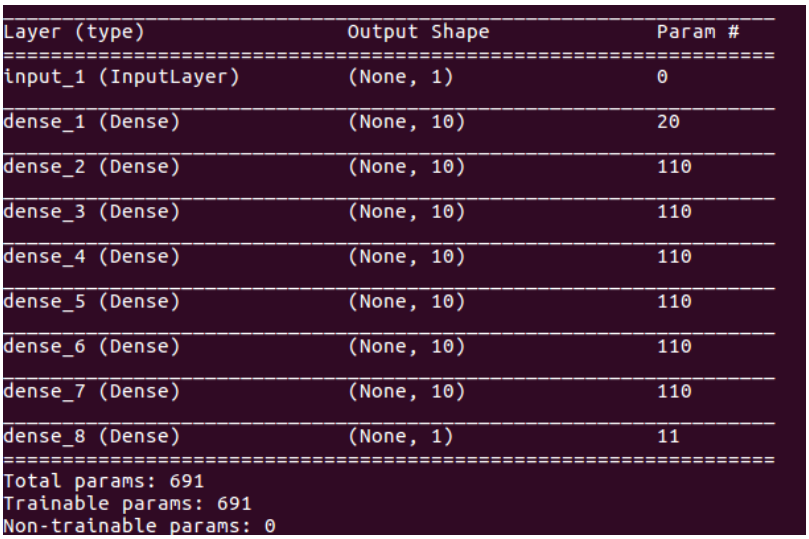
shallow



Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 1)	0
dense_9 (Dense)	(None, 16)	32
dense_10 (Dense)	(None, 12)	204
dense_11 (Dense)	(None, 13)	169
dense_12 (Dense)	(None, 19)	266
dense_13 (Dense)	(None, 1)	20

Total params: 691
Trainable params: 691
Non-trainable params: 0

Middle



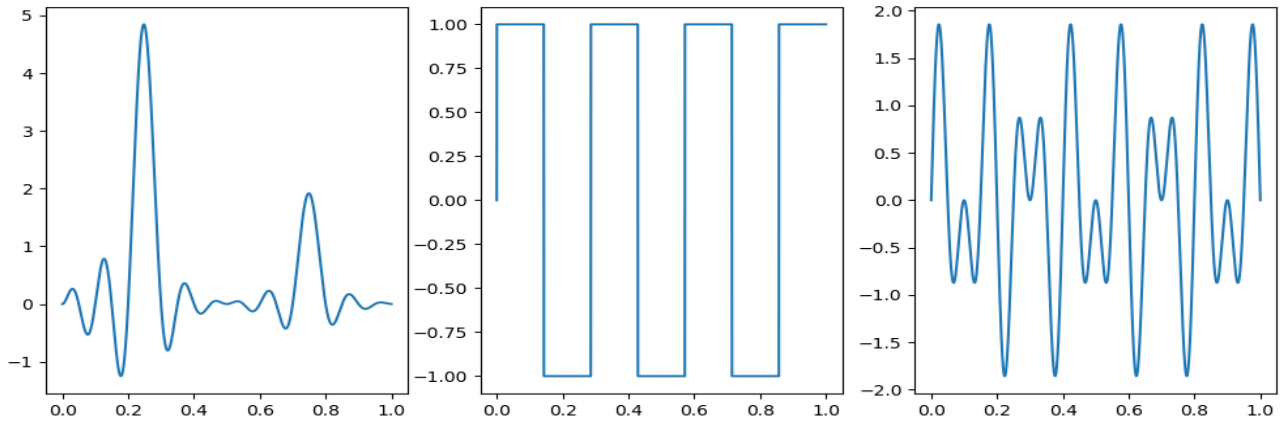
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1)	0
dense_1 (Dense)	(None, 10)	20
dense_2 (Dense)	(None, 10)	110
dense_3 (Dense)	(None, 10)	110
dense_4 (Dense)	(None, 10)	110
dense_5 (Dense)	(None, 10)	110
dense_6 (Dense)	(None, 10)	110
dense_7 (Dense)	(None, 10)	110
dense_8 (Dense)	(None, 1)	11

Total params: 691
Trainable params: 691
Non-trainable params: 0

Deep

使用的 functiong 上，共訓練了三種不同的 function，分別是

$y = \frac{\sin(20\pi x) \times \tan(2\pi x)}{(2\pi x + 0.5)}$、 $y = \text{sign}(\sin(7\pi x))$、 $y = 2 \times \sin(20\pi x) \times \cos(5\pi x)$

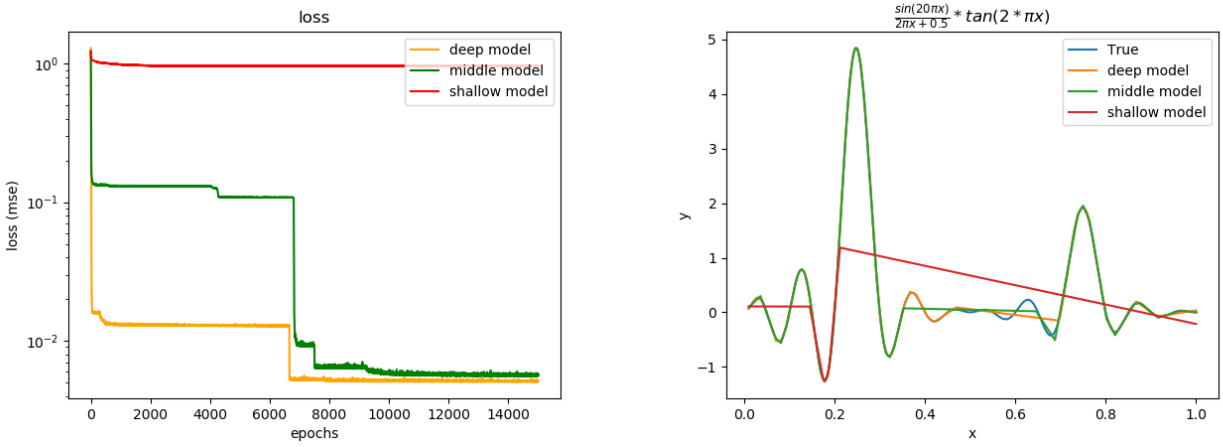


故意挑選比較複雜的函式，讓淺和深的模型在 fit function 的時候能讓淺和深的模型差別更大。

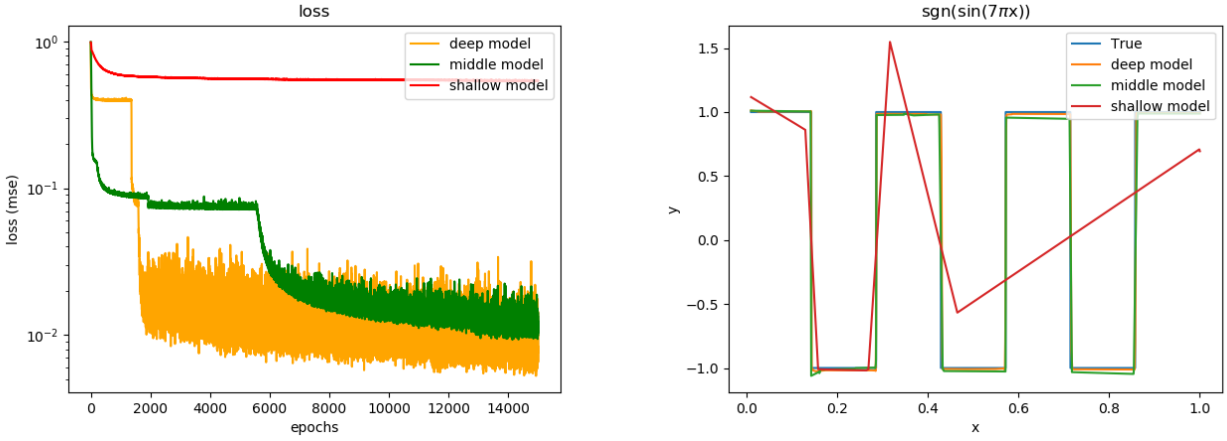
- **In one chart, plot the training loss of all models. (0.5%)**
- **In one graph, plot the predicted function curve of all models and the ground-truth function curve. (0.5%)**

左邊的圖分別呈現不同 function 的 loss，紅線代表是 shallow、綠色是 middle、橘色是 deep，右邊的圖呈現的是 ground-truth function curve 和三個 model 的 predict 出來的 curve，藍色線是 ground-truth cuve、紅色線是 shallow、綠色是 middle、橘色是 deep model。

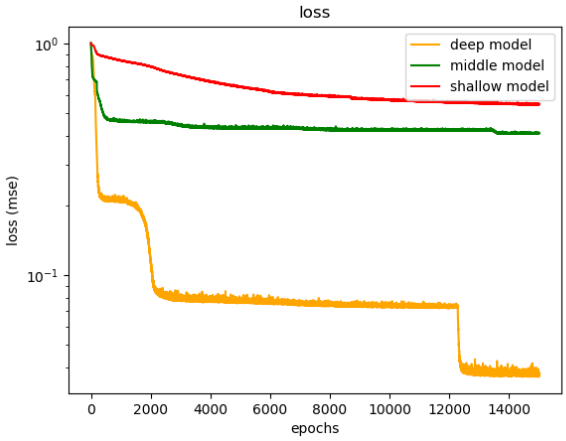
function1 loss & curves：

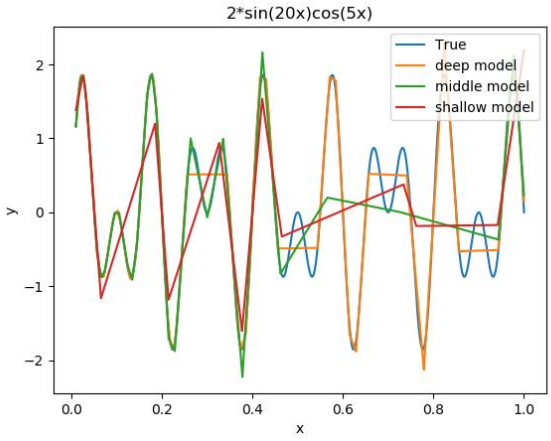


function2 loss & curves：



function3 loss & curves：





- **Comment on your results. (1%)**

  從上述三個 function 的訓練過程中可以看出雖然整體參數量是一樣(691 個)的但是 loss 的情形卻差很多，可以看到紅色線的 loss 都比其他兩者來的大許多，而且隨著 epoch 數訓練越大 loss 並沒有下降，可知較淺較寬的 model 預測結果是差的。比較 deep 和 middle 的模型，在前兩個 function 中它們的 loss 是比較相近的，兩者在 fit 前面兩個 function 結果是相近的。而第三個 function 的 curve 有非常多的起伏，也是比前面兩個函數較為複雜，在此時可以看到 deep 的結果是比 middle 好的，所以總體而言模型越深是可以 fit 越複雜的 function，且 performance 也比較好。

  除此之外，從 loss 的圖上可以看到雖然 deep 模型可以得到比較低的 loss，但是 loss 的下降速度卻不一定比較快，從第二張 loss 可以看到，在前面幾個 epoch 的時候 middle 的模型 loss 反而較先降下去。這代表雖然 deep 的模型 loss 可以下降到很低但不代表它下降的速度比較快。

- **Use more than two models in all previous questions. (bonus 0.25%)**
- **Use more than one function. (bonus 0.25%)**

  已合併在上面

1-1-2: Train on actual task

- **Describe the models you use and the task you chose. (0.5%)**

  Task 選擇 MNIST 手寫數字辨識，訓練在三個不同深度，參數量差不多的 DNN 上面

  BATCH_SIZE : 64, EPOCHS : 100, learning_rate : 0.001, optimizer: SGD

  淺層 DNN

  Net_DNN(

  (fc1): Linear(in_features=784, out_features=20, bias=True)

  (fc2): Linear(in_features=20, out_features=10, bias=True)

  (fc3): Linear(in_features=10, out_features=10, bias=True))

  參數量 ：16020

  中層 DNN

  Net_DNN(

  (fc1): Linear(in_features=784, out_features=18, bias=True)

  (fc2): Linear(in_features=18, out_features=26, bias=True)

  (fc3): Linear(in_features=26, out_features=24, bias=True)

(fc4): Linear(in_features=24, out_features=22, bias=True)

(fc5): Linear(in_features=22, out_features=10, bias=True))

參數量 ：16052

深層 DNN

Net_DNN(

(fc1): Linear(in_features=784, out_features=17, bias=True)

(fc2): Linear(in_features=17, out_features=28, bias=True)

(fc3): Linear(in_features=28, out_features=26, bias=True)

(fc4): Linear(in_features=26, out_features=22, bias=True)

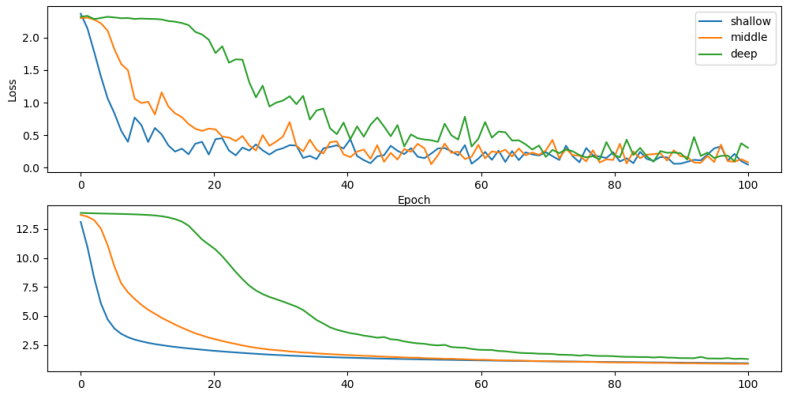
(fc5): Linear(in_features=22, out_features=18, bias=True)

(fc6): Linear(in_features=18, out_features=16, bias=True)

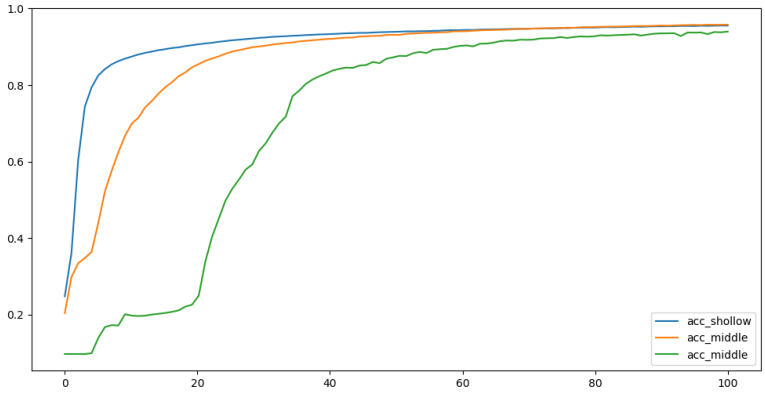
(fc7): Linear(in_features=16, out_features=10, bias=True))

參數量 ：16085

- **In one chart, plot the training loss of all models. (0.5%)**



- **In one chart, plot the training accuracy. (0.5%)**

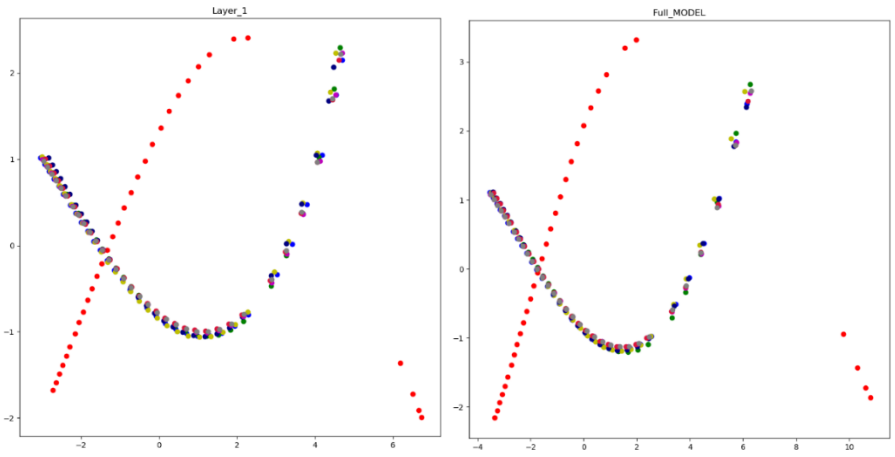


- **Comment on your results. (1%)**

LOSS 圖之上圖為每個 EPOCH 取 3 個點繪出之 LOSS，下圖為每個 EPOCH 最後一個點之 LOSS，ACCURACY 之圖作法為每個 EPOCH 訓練過後測試整個 TRAINING SET 得出之正確率。

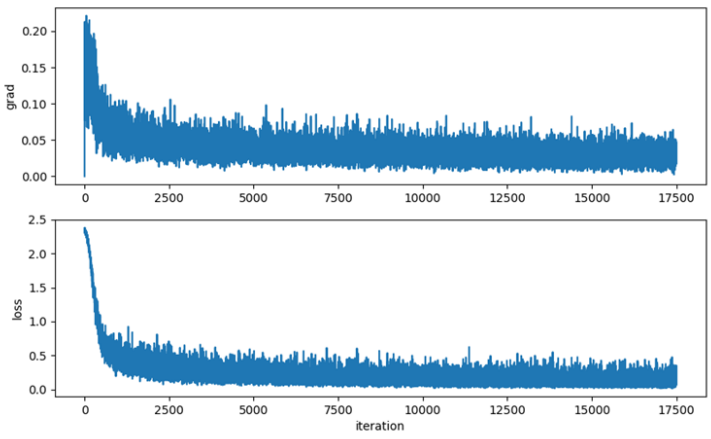
而觀察發現，在模式設定此狀況下(參數量約 16000)，深的 MODEL 之表現是較為不好的，反而淺層的 MODEL 之 LOSS 下降較快(ACC 上升較快)，但其實在大約 95 個 EPOCH 之後，中等深度的 MODEL 之表現有超過淺層的 MODEL(LOSS 低於淺層 MODEL、ACCURACY 高於淺層 MODEL)，但是深層的 MODEL 在這個參數量下的表現是最差的，不僅收斂速度慢，且表現也沒有比較好。

- **1-2-1: Visualize the optimization process**
  - **Describe your experiment settings. (1%)**
    DNN train on MNIST.
    BATCH_SIZE : 64, EPOCHS : 100, learning_rate : 0.001, optimizer: SGD
    Net_DNN(
    (fc1): Linear(in_features=784, out_features=20, bias=True)
    (fc2): Linear(in_features=20, out_features=10, bias=True)
    (fc3): Linear(in_features=10, out_features=10, bias=True))
    參數量 ：16020
    使用 PCA 降維。
  - **Train the model for 8 times, selecting the parameters of any one layer and whole model and plot them on the figures separately. (1%)**



  - **Comment on your result. (1%)**
    每次訓練用不同的顏色表示，發現在沒有打亂訓練集的狀況下，即使不同的 random initial 參數，模式也很難收斂到其他的地方，不同於助教所展示的。而紅色部分是將一開始的參數調在跟其他訓練集很不一樣的地方，因此稍微有不一樣的表現。

- **1-2-2: Observe gradient norm during training.**
  - **Plot one figure which contain gradient norm to iterations and the loss to iterations. (1%)**



- **Comment your results. (1%)**

  隨著 gradient norm 的下降 loss 也隨之下降，推測是跟弘毅老師講的一樣，在梯度下降的過程中，隨著一開始在梯度大的地方往下走，最後到了平坦的地方，梯度小、loss 也很小，但可能因為用的方法有加動量、沒有做 feature scaling 而導致最後震盪很大，並沒有很穩定的收斂到一個地方，但是大致上的趨勢還是正確的，看起來有收斂到一個梯度較小的區域。

1-2-3: What happens when gradient is almost zero?

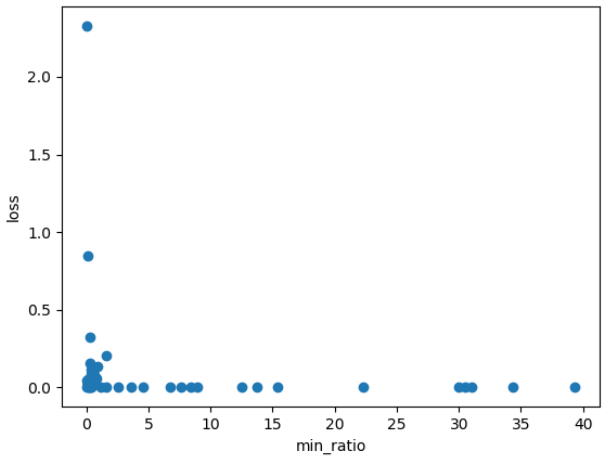
- **State how you get the weight which gradient norm is zeros and how you define the minimal ratio.**

  使用牛頓法近似去逼近很小的 gradient norm。

  Minimal ratio:

  計算 hessian matrix，並找到它的 eigenvalues。minimal ratio 則是大於 0 的 eigenvalues 數量佔整個 eigenvalues 的比例。

- **Train the model for 100 times. Plot the figure of minimal ratio to the loss. (2%)**



- **Comment your result. (1%)**

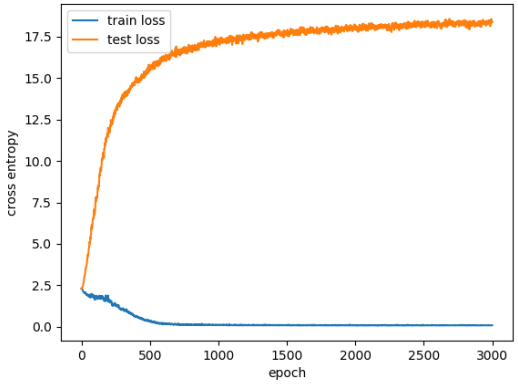
  牛頓法在這個訓練集裡有些不穩定，有時候會產生爆掉的情況 XD，但大部分的情況集中在左下角也就是穩定收斂的狀況。

1-3-1: Can network fit random variables?

- **Describe your settings of the experiments. (e.g. which task, learning rate, optimizer) (1%)**

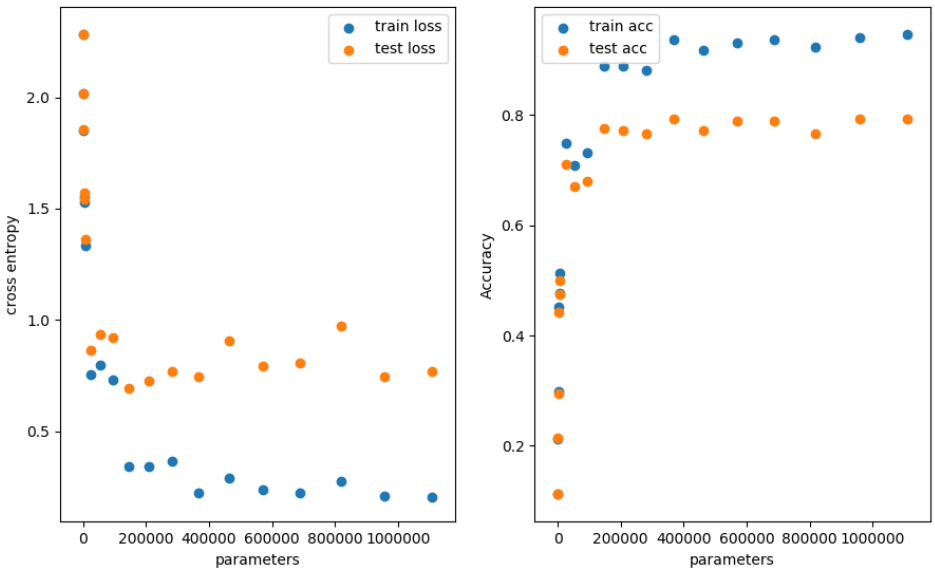
  使用的是 mnist 的 task，兩層的 network(不包括 input 和 output layer)，每層分別是 512、128 個 units，batch size 是 512，epochs 是 3000，使用的 optimizer 是 Adam 而初始的 learning rate 為 0.001。

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



- **1-3-2: Number of parameters v.s. Generalization**
  - **Describe your settings of the experiments. (1%)**

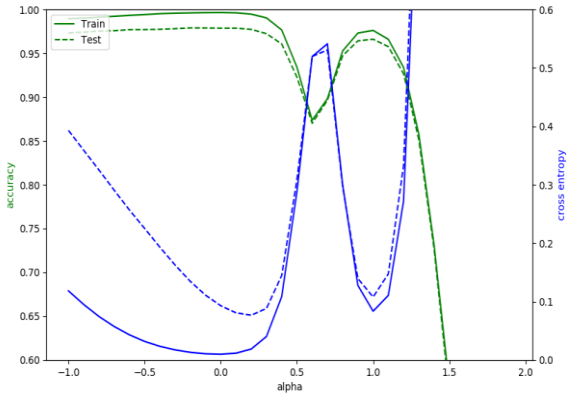
    此次實驗用的是 cifar-10 的 task，使用的是 CNN 的 network，總共有三層 CNN 和 Maxpooling，最後兩層全連接層，共訓練了 15 個結構相同參數量不同的模型，參數最少從 121 到最多 110 萬，使用的 optimizer 一樣是 Adam，每個模型都訓練 100 epochs。
  - **Plot the figures of both training and testing, loss and accuracy to the number of parameters. (1%)**



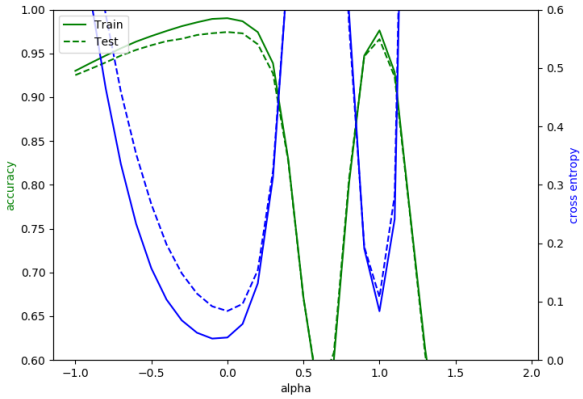
- **1-3-3: Flatness v.s. generalization**
  - **Part 1:**
    - **Describe the settings of the experiments (e.g. which task, what training approaches) (0.5%)**

      使用的 task 是 mnist，network 使用兩層全連接層，optimizer 使用了 Adam，分別 train 了 batch size 為 64 和 2048 的結果。同時也有 train 同樣 batch size 但分別使用 adagrad 和 adam 的 optimizer 的 model。
    - **Plot the figures of both training and testing, loss and accuracy to the number of interpolation ratio. (1%)**

      左邊是 batch size 比較，右邊是 adagrad 和 adam 的比較



alpha=0 代表 SB，alpha=1 代表 LB



alpha=0 代表 adagrad，alpha=1 代表 adam

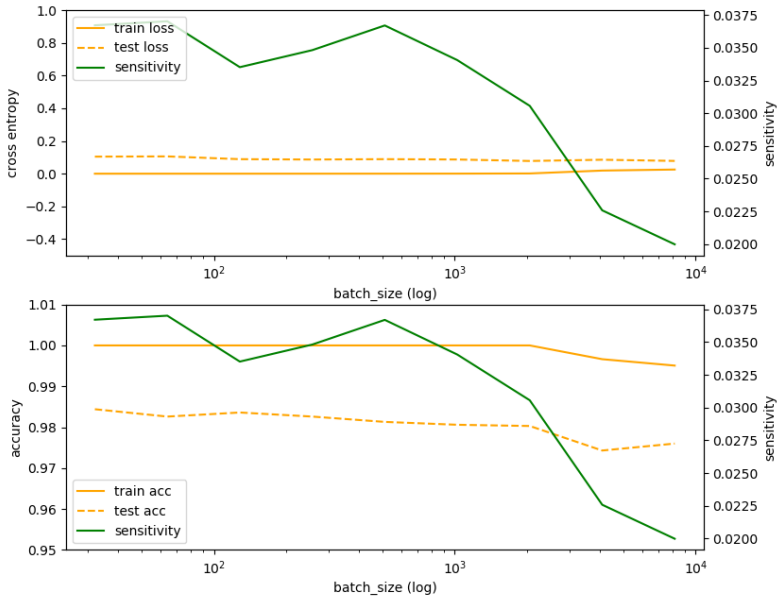
- **Comment your result. (1%)**

  左圖中可以看到 SB 找到的 local minimum 比較平坦，而 LB 則比較陡，而右圖則可以看到使用 adagrad optimizer 時找到的 local minimum 比 adam 來的平坦許多。而比較兩圖時可以發現，使用不同 batch size 得到的 local minimum 之間的 loss(左圖 alpha 大約為 0.5 時的 loss)比右圖的來的低(大約 alpha 0.5 之處的 loss)，這似乎和老師上課提到集水區概念是有關連的，使用不同的 optimizer 會走到不一樣的"集水區"，所以他們之間的分水嶺會比較高(如右圖)，但是使用相同的 optimizer 只是 batch size 不同可能找到的 local minimum 會在同個"集水區"，而他們之間的分隔可能就只是一座小山。

- **Part 2:**
  - **Describe the settings of the experiments (e.g. which task, what training approaches) (0.5%)**

    使用的 task 是 mnist，network 使用兩層全連接層，optimizer 使用了 Adam，並將模型訓練到 loss 幾乎為 0。Sensitivity 的算法是先算 loss 對 input data 的 gradient，並將得到的 gradient 全部平方相加起來在開根號得到 sensitivity。

  - **Plot the figures of both training and testing, loss and accuracy, sensitivity to your chosen variable. (1%)**

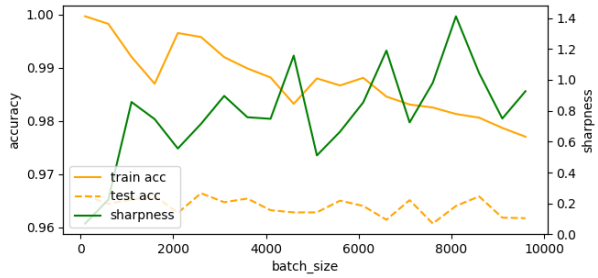


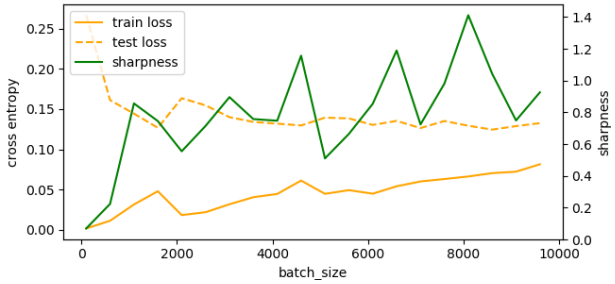
  - **Comment your result. (1%)**

    此結果為將 testing data 放到 input data 之中的結果。可以看到 sensitivity 隨 batch size 越大越低，跟助教在 1-3 的投影片相同。此結果跟原本我的預期是有點差距的，原以為 batch size 越大 generalization 越差(根據上課時講 Sharpness 提到的結果)，所以 sensitivity 會越大，但是這邊的結果卻不太一樣，推測可能 sensitivity 跟 sharpness 沒有直接的關係，而且 sensitivity 是比較 heuristic 的算法所致。

- **Bonus: Use other metrics or methods to evaluate a model's ability to generalize and concretely describe it and comment your results.**

  在這邊使用了助教投影片中 Sharpness 的算法來計算在不同 batch size 中 sharpness 的大小。使用的 task 依然是 mnist，但是為了簡化運算空間以及時間所以將 network 大小調小，兩層全連接層分別都是 32 個 units，training loss 都壓在 0.1 之下。從圖中可以看到雖然 sharpness 震動幅度蠻大的，但是可以看到隨著 batch size 越大 sharpness 是有漸漸變大的，這結果跟上課時老師提到的是一致的。





分工：

羅章碩：1-1-1、1-3 全

鄒適文：1-1-2、1-2 全