- 1-1

1. **Simulate a Function**

Describe the models you use, including the number of parameters and the function you use.
我們使用了三種模型，三種都是使用sigmoid作為activate function，其中第一個模型為 Shallow Model，共有 526 個參數；第二個模型為 3 Layer Dense Model，共有526個參數；第三個模型為 6 Layer Dense Model，共有527個參數，這三個模型的架構如下。

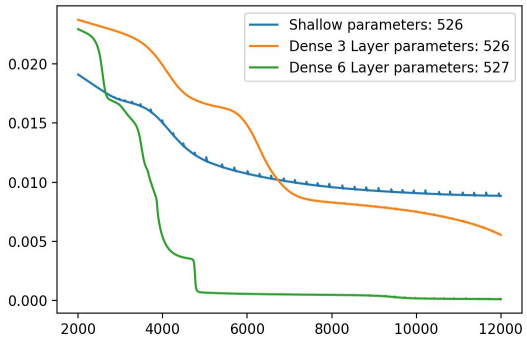
```
Shallow
Model_0(
  (fcn1): Linear(in_features=1, out_features=175, bias=True)
  (fcn2): Linear(in_features=175, out_features=1, bias=True)
)
----------------------------------------------------------------
Dense 3 Layer
Model_1(
  (fcn1): Linear(in_features=1, out_features=21, bias=True)
  (fcn2): Linear(in_features=21, out_features=21, bias=True)
  (fcn3): Linear(in_features=21, out_features=1, bias=True)
)
----------------------------------------------------------------
Dense 6 Layer
Model_2(
  (fcn1): Linear(in_features=1, out_features=11, bias=True)
  (fcn2): Linear(in_features=11, out_features=11, bias=True)
  (fcn3): Linear(in_features=11, out_features=11, bias=True)
  (fcn4): Linear(in_features=11, out_features=10, bias=True)
  (fcn5): Linear(in_features=10, out_features=10, bias=True)
  (fcn6): Linear(in_features=10, out_features=1, bias=True)
)
----------------------------------------------------------------
```

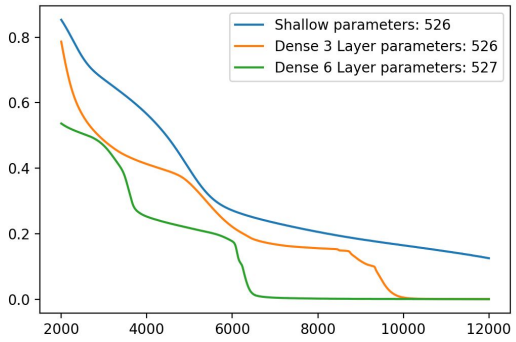
而我們所比較的函式有兩個，分別是：

1. $\frac{e^{sin(3x)+cos(2\pi x)}}{2\pi} + sin(3\pi)$ 以及 2. $sin(3x) + cos(2\pi x) + e^{sin(2x)}$

In one chart, plot the training loss of all models.

1. 2.

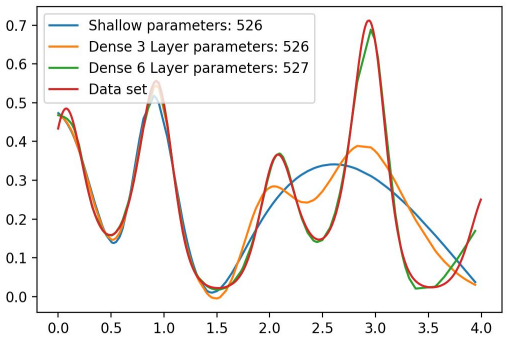


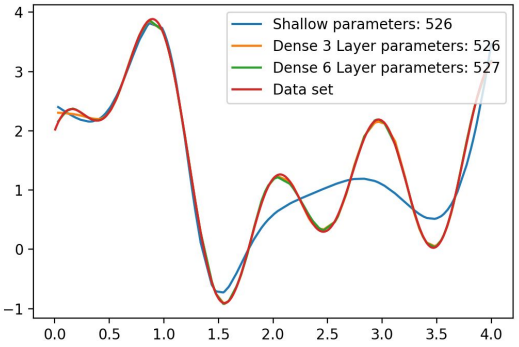


藍色為 shallow model 的 training loss，橘色為 3 layer dense model 的 training loss，綠色則為 6 layer dense model 的 training loss 。

In one graph, plot the predicted function curve of all models and the ground-truth function curve.

1. 2.





紅色為正確的函數方程式。

Comment on your results.

從上面的四張圖中，可以看出雖然參數數量幾乎一樣，但是擁有較深結構的模型， training loss 會下降至比較淺的結構低。會造成這樣的原因，我們推測是因為多層結構的模型，每一層的參數會互相影響，將參數的效果放大；而不會像單層結構的模型，每個參數間不會互相影響，只擁有單一的效果。因此，從和正確方程式比較的圖形來看，也會發現，多層結構的模型訓練出來的方程式會與正確方程式十分接近，而單層結構的模型，雖然擁有一樣的參數量，卻不會和正確的方程式相似。

2. Train on Actual Tasks

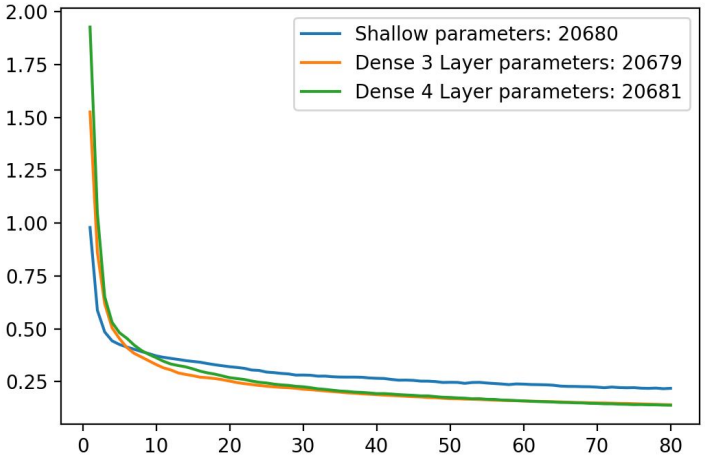
Describe the models you use and the task you chose.

我們使用了三種模型，三種都是使relu作為activate function，其中第一個模型為 Shallow Model，共有 20680 個參數；第二個模型為 3 Layer Dense Model，共有 20679 個參數；第三個模型為 4 Layer Dense Model，共有 20681 個參數，這三個模型的架構如下。

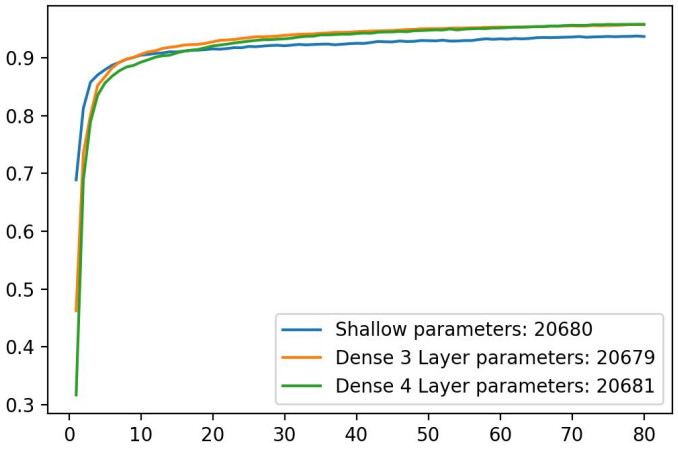
```
Shallow
Model_0(
  (fcn1): Linear(in_features=784, out_features=26, bias=True)
  (fcn2): Linear(in_features=26, out_features=10, bias=True)
)
----------------------------------------------------------------
Dense 3 Layer
Model_1(
  (fcn1): Linear(in_features=784, out_features=25, bias=True)
  (fcn2): Linear(in_features=25, out_features=29, bias=True)
  (fcn3): Linear(in_features=29, out_features=10, bias=True)
)
----------------------------------------------------------------
Dense 4 Layer
Model_2(
  (fcn1): Linear(in_features=784, out_features=24, bias=True)
  (fcn2): Linear(in_features=24, out_features=28, bias=True)
  (fcn3): Linear(in_features=28, out_features=29, bias=True)
  (fcn4): Linear(in_features=29, out_features=10, bias=True)
)
----------------------------------------------------------------
```

而我們所做的 Task 是 MNIST。

In one chart, plot the training loss of all models.



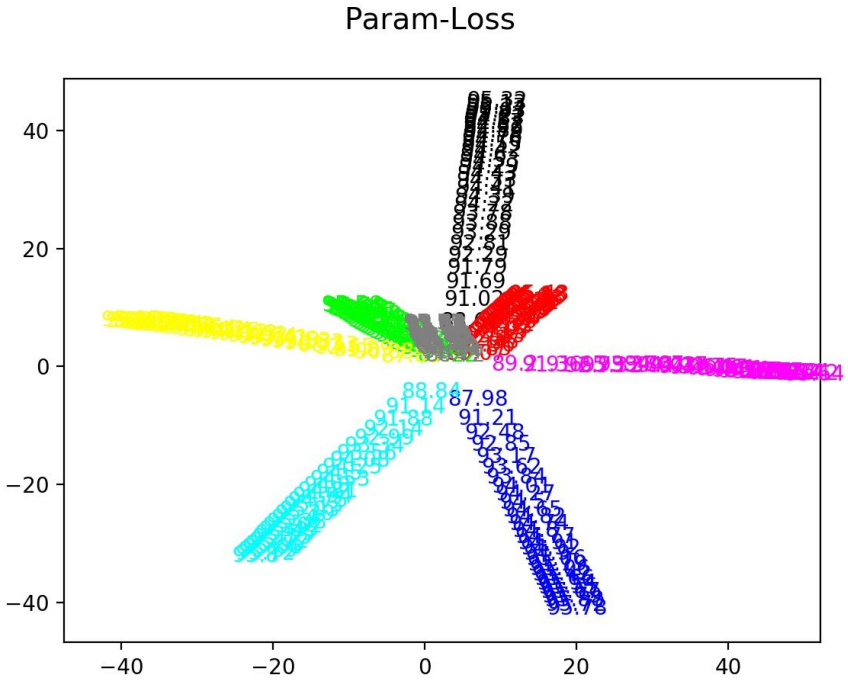
In one chart, plot the training accuracy.



Comment on your results.

從上面的兩張圖會發現即使是在實際的任務中，單層模型的表現雖然一開始 loss 和 accuracy 會比多層模型好。但是隨著 epoch 增加，多層模型的表現會隨之上升，而超越單層模型，而這也如同我們前述的猜測，多層結構的模型，每一層的參數會互相影響，將參數的效果放大；而不會像單層結構的模型，每個參數間不會互相影響，只擁有單一的效果。

- 1-2
  1. **Visualize the Optimization Process**
     Training Task：MNIST database of handwritten digits
     Training Set：60000張手寫數字圖片(大小28x28)
     Network 架構：先將 28x28 的圖片攤平成 784 的一維矩陣，經過一層 50 維的 Linear Layer 後，最後輸出 10 維的 softmax 結果。Activation 都用 relu，optimizer 用 Adagrad。
     做法：總共訓練 8 次相同的 network，每 3 個 epoch 將 network 中的 parameter 全部攤平後記錄 weights 以及 accuracy，最後將 weights 以 PCA 降成 2 維並將對應的 accuracy scatter 到 2 維坐標軸上，結果為下圖：

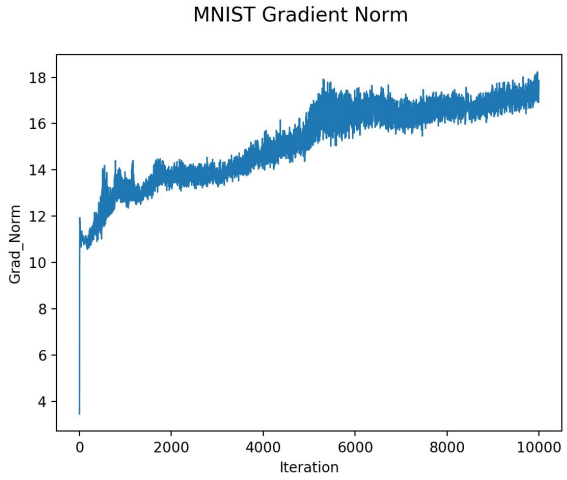


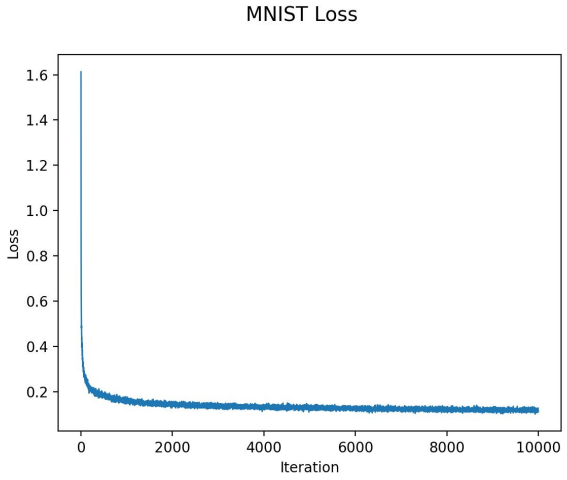
     可以看到黑、黃、淺藍、深藍、桃紅色的訓練過程大致上離開中心的速度較快，而綠、灰、紅則較慢，可以知道訓練方向的不同。

  2. **Observe gradient norm during training**
     我們分別做了兩種 case，分別為 MNIST 以及 Simple function：
     - MNIST
       Training Set：60000 張手寫數字圖片(大小28x28)
       Network架構：先將 28x28 的圖片攤平成 784 的一維矩陣，經過一層 10 維的 Linear Laye r後，最後輸出 10 維的 softmax 結果。Activation 都用 relu，optimizer 用 Adagrad。
     - Simple function - $e^{(sin(\pi x)+cos(3\pi x))/2}+sin(2x)$
       Training Set：從 function 在x=[0, 4]中隨機取樣 1000 個點
       Network 架構：將 input 經過兩層 10 維的 Linear Layer 後，最後輸出1維的結果。Activation 都用 sigmoid，optimizer 用 Adam。

     結果為下面四張圖：
     - MNIST

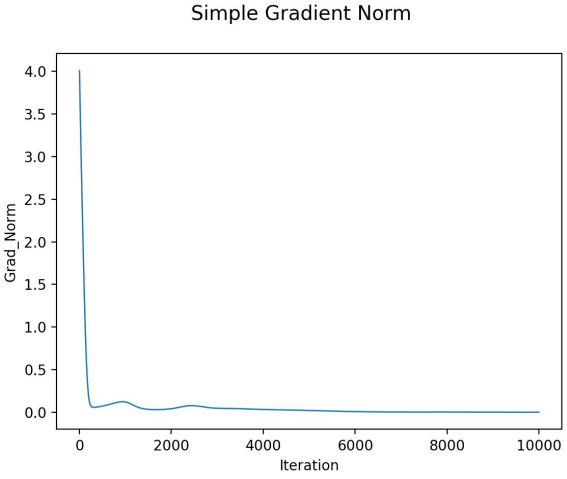
可以看到在 MNIST 的訓練過程中，Loss 雖然有上下起伏，但是整體趨勢是往下的。而在 gradient-norm 中起伏較大，整體趨勢是往上。

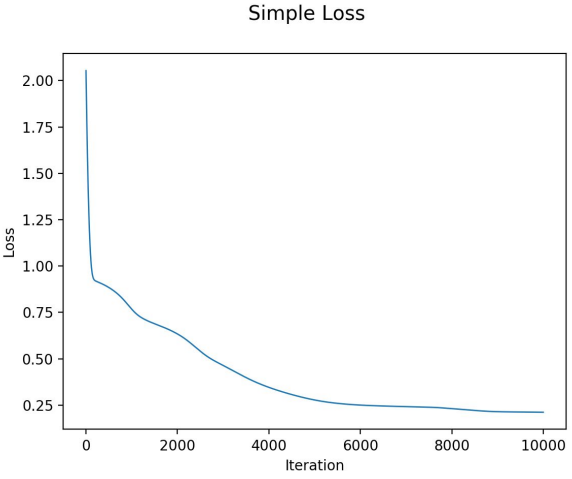




- Simple function - $e^{(sin(\pi x)+cos(3\pi x))/2}+sin(2x)$

  可以看到在 Simple function 的訓練過程，基本上 gradient-norm 以及 loss 都是往下的趨勢。loss 在約 200 iteration 的部分變得較緩主要是因為 gradient-norm 已經趨近於0因此更新參數較緩慢。而可能是因為 function 較 MNIST 簡單，容易 fit 的關係，訓練過程並沒有如 MNIST 上下抖動的現象(更平滑)。

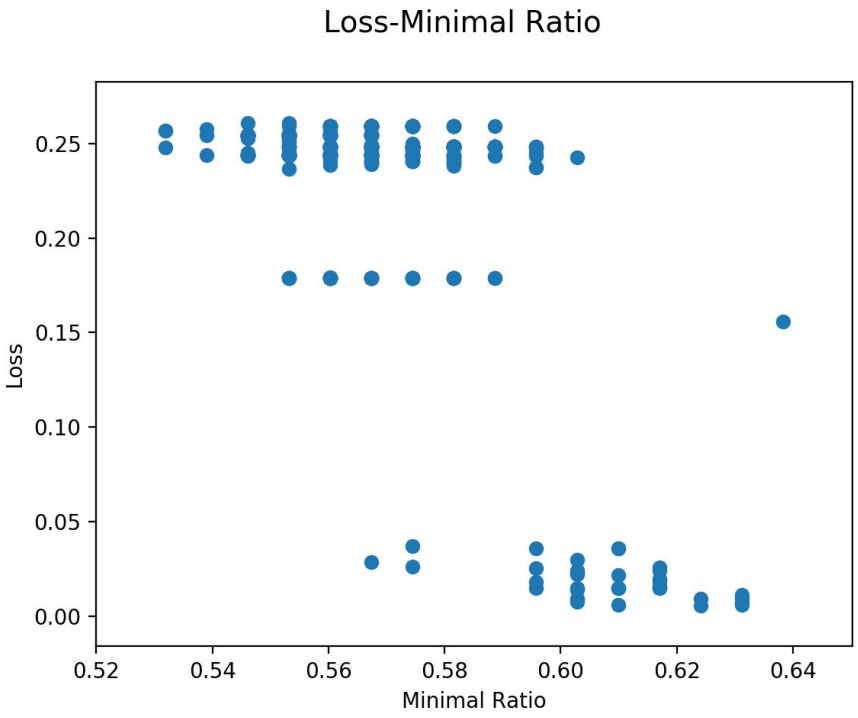




3. What happens when gradient is almost zero

採用的 case 也是 Simple function - $e^{(sin(\pi x)+cos(3\pi x))/2}+sin(2x)$ ，和 1-2.2 相同架構。

方法：在相同的 network 訓練 100 次，在訓練時先以原本的 loss 當作目標函數，每一個 epoch 算出 gradient-norm，當 gradient-norm 小於 0.0025 時，將目標函數換為 gradient-norm，並且繼續訓練。在目標函數換為 gradient-norm 後，繼續跑 1000 個 epoch，training 時若 gradient-norm 小於 0.003 時，計算當下 model 的 hessian 值，並且利用 numpy 的 eigvals 算出 hessian 矩陣的 eigen values，並且算出 eigen values 中正數的比率，最後畫到圖上，結果如下圖所示：



可以看到大致上分為三個區域：loss 分別約為 0.25、0.17、0.02 的情況，大致上的情形為 loss 越低，minimal ratio 會越高，表示越像 local minimun。在這三個 loss 以外的範圍沒有點，表示在這三個 loss 的部分都是較平坦的部分，但是因為 minimal ratio 都只有0.52 ~ 0.64，因此可以說這些部分大致上都是 saddle point。

- 1-3

1. **Can network fit random variables?**

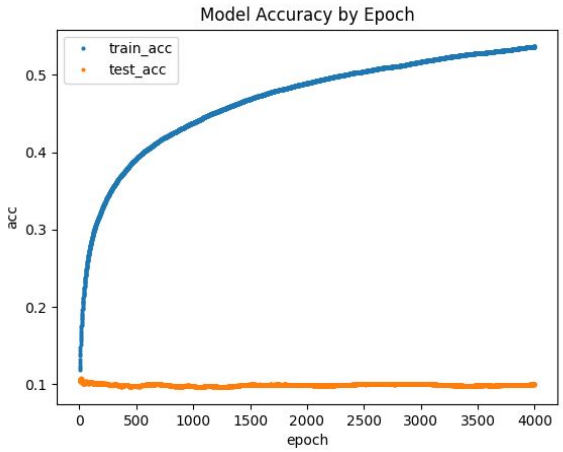
使用 MNIST 資料集並以 如下 NN 的架構進行實驗。

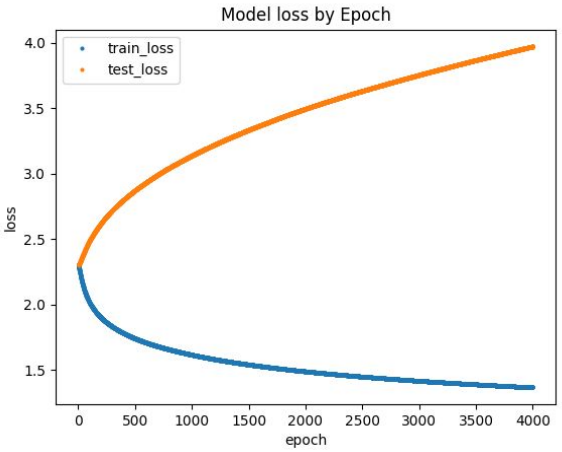
Network 架構:

(a) 兩層 Linear Layer 參數量: 203530

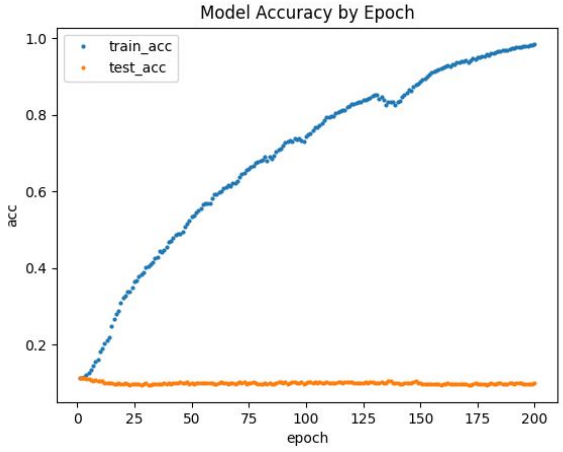
(b) 六層 Linear Layer 參數量: 5654538

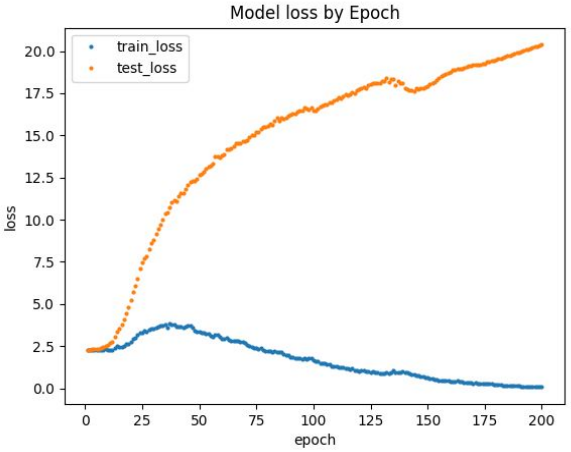
因為一開始不知道對於打亂答案的資料集訓練的難度如何，所以先用較為簡單的模型 (a) 進行測試。訓練過程如下二圖:





在經過 4000 個 Epoch (CPU 約時 60 分鐘) 的訓練後，訓練集的準確率達到 0.5 左右。而隨著訓練的次數越多 訓練的 Loss 持續下降至 0.6 但測試的 Loss 上升約至 4.0。雖然這個簡單的模型的確能 fit 隨機打亂答案的資料集，但是訓練的準確率到 0.4 以後上升幅度就漸漸趨緩了。所以在模型 (b) 我們加深模型並增加參數量，且使用同樣的 batch size 進行訓練。訓練過程如下二圖:

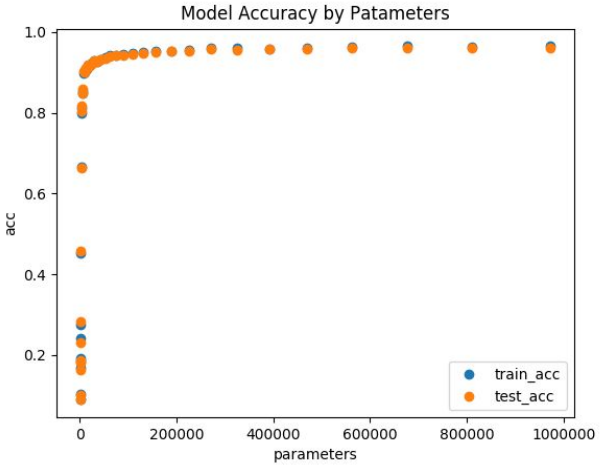


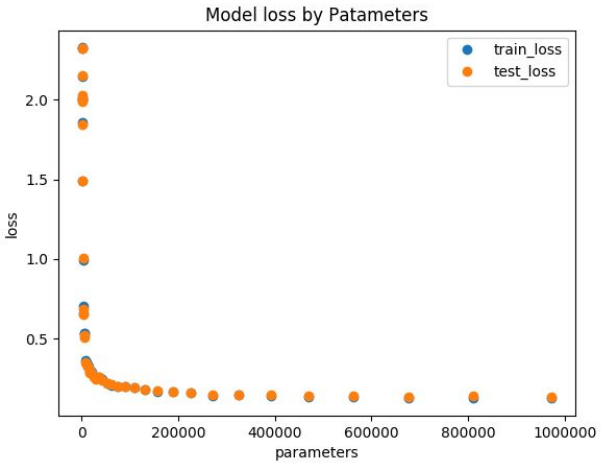


模型 (b) 在經過 200 個 Epoch 的訓練後 (CPU 約時 80 分鐘)，訓練的準確率來到 0.98325，我們可以觀察到 NN 的模型確實可以硬記下來所有的答案，但是很顯然這對於測試的準確率一點幫助都沒有，而模型 (b) 的訓練 Loss 最後達到 0.000216 ，測試的 Loss 反而上升到 20 左右，顯然隨著輸出的結果越貼近訓練的分佈，反而會越遠離測試結果的分佈。此外我們也確認了在這個任務中加深並增加參數量，的確可以幫助模型的訓練。

2. Number of parameters vs Generalization

使用 MNIST 資料集並以 35 個不同的 NN 的架構進行實驗，35 個模型都是 2 層的模型，但是我們使模型的 hidden size 指數上升，參數量並進行觀察。訓練時使用的資料大小是 60000 筆，並設定 batch size 的大小為 256，訓練 1 個 epoch。





可以觀察到在參數量小於 2 萬時，這樣的設定不能訓練出準確的模型，但隨著模型參數量增大至 5 萬，訓練與測試的準確率都來到 0.9 左右，再一路加大 hidden size 的大小，使的最後參數量達到 97 萬。對於 MNIST 的資料集來說，97 萬的參數量是個相對複雜的模型，但使用 97 萬參數量的模型相對於其他的並沒有出現 overfitting 的現象。我們認為一方面是因為 Mnist 是相對簡單的訓練集，另一方也顯示了對於越複雜的 NN 模型，從結果上來看，overfitting 的程度並不會越大。

3. Flatness v.s. Generalizatio

Part 1:

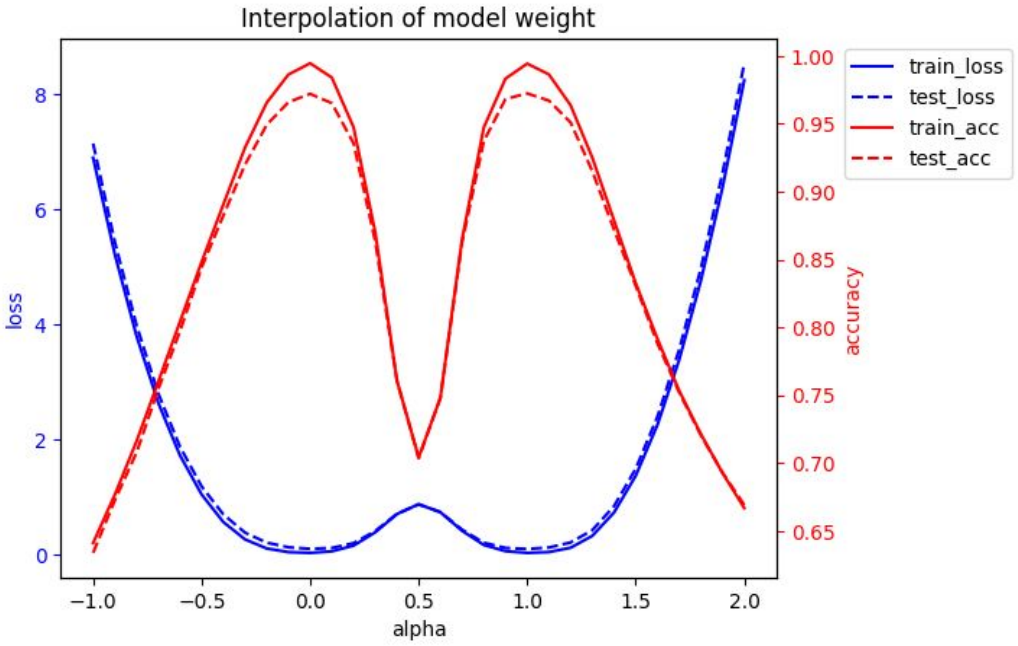
第一個部分的目標是觀察參數在某一個方向上的變化與 loss 之間的關係。

我們使用 MNIST 資料集，模型部分使用 2 層的 Linear layer，hidden size 為 50。

使用不同的 batch size 大小 - 64 與 1024 分別經過 200 與 250 個 Epoch 的訓練後，batch size 為 64 的訓練下測試的 loss 為 0.0966 而在 batch size 為 1024 的訓練下則是 0.0939，測試準確率亦十分接近。接著以這兩個模型參數的內外插做為新模型的參數，公式如下(同投影片做法):

$$\boldsymbol{\theta_\alpha} = (1 - \alpha)\,\boldsymbol{\theta_1} + \alpha\boldsymbol{\theta_2}$$

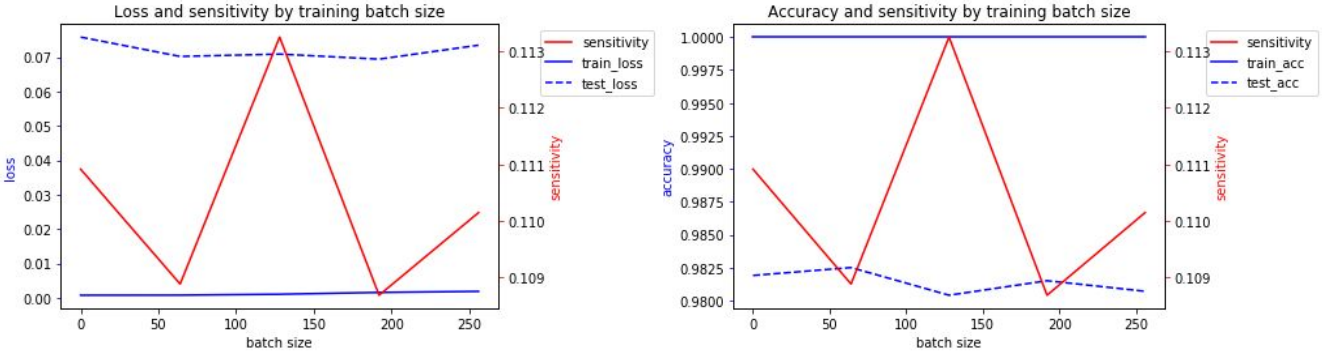
做出來的圖形如下圖:



可以看出似乎找到了兩個不同的解，而在兩個解之間移動的過程中，[0.0, 0.5] 與 [1.0, 0.5] 的兩個區段間 ，loss 有很明顯的上升，顯示兩個解之間的 error surface 有類似分水嶺的狀況存在，但是兩個解附近 error suface 從這個維度看不出差異。

Part 2:
使用 MNIST 資料集，以 2 層的 Linear Layer 並設 hidden size 為 256 調整訓練的 batch size 以進行實驗。目標是觀察 Accuracy、Loss 以及 Sensitivity 之間的關係。而實驗的 5 個 batch size 為 [64, 128, 256, 512, 1024] ，訓練 200 個 Epoch。

計算 10000 筆測試資料的 Sensitivity 後發現，對於不同的資料點對於不同的模型的偏好度不一，有些在小的 epoch size 上表現的比較好，有些則不然。所以我們認為在這樣的訓練條件下，無法看出 batch size 與 sensitivity 的關聯性，有可能是這 5 組參數的表現真的差不多。也因為這樣我們決定改畫 1000 筆測試資料平均的 Sensitivity，結果如下二圖。



從結果可以看出來，不是 epoch 越小就表現得越好。此外，Loss 較低、Accuracy 較高的參數 Sensitivity 也就較低，這個結果不太出人意料之外，因為 Sensitivity 是 Loss 對測試資料算出的 gradient norm。