HW1

組員： b03902093 張庭維
b03902101 楊力權
b03902102 廖廷浩

1-1(with BONUS 5, 6)

Simulate a Function:

1. Describe the models you use, including the number of parameters (at least two models) and the function you use. (0.5%)

model1

fc1:Dense	input:	1
	output:	10
fc2:Dense	input:	10
	output:	27
fc3:Dense	input:	27
	output:	10
fc4:Dense	input:	10
	output:	1

參數總量：608

model2

fc1:Dense	input:	1
	output:	200
fc2:Dense	input:	200
	output:	1

參數總量：601

model3

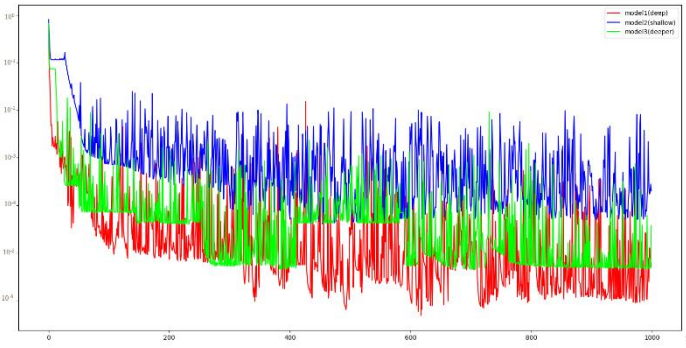
fc1:Dense	input:	1
	output:	8
fc2:Dense	input:	8
	output:	16
fc3:Dense	input:	16
	output:	16
fc4:Dense	input:	16
	output:	8
fc5:Dense	input:	8
	output:	4
fc6:Dense	input:	4
	output:	1

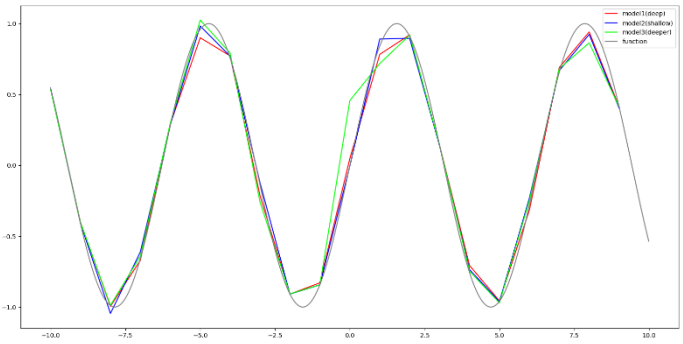
參數總量：609

The function we use: $f(x) = \sin x$

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

2. 3. In one graph, plot the predicted function curve of all models and the ground-truth function curve and the loss of two models. (1%)





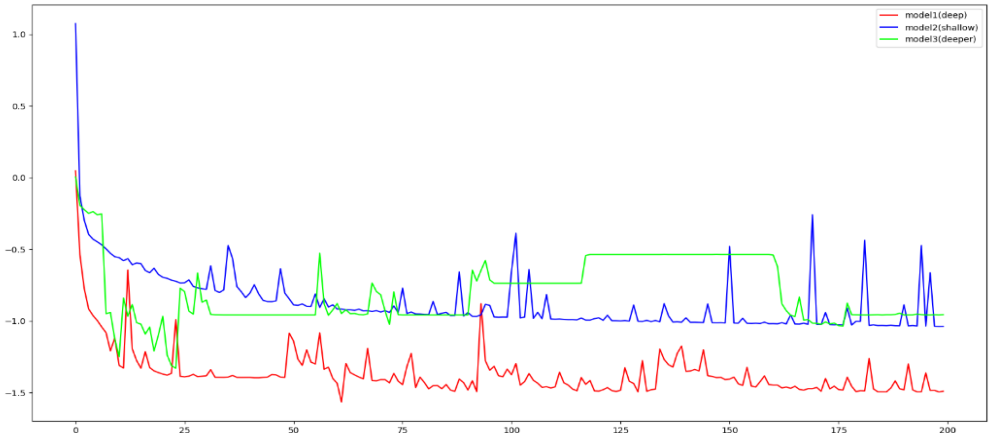
紅色: model1 藍色: model2 綠色: model3

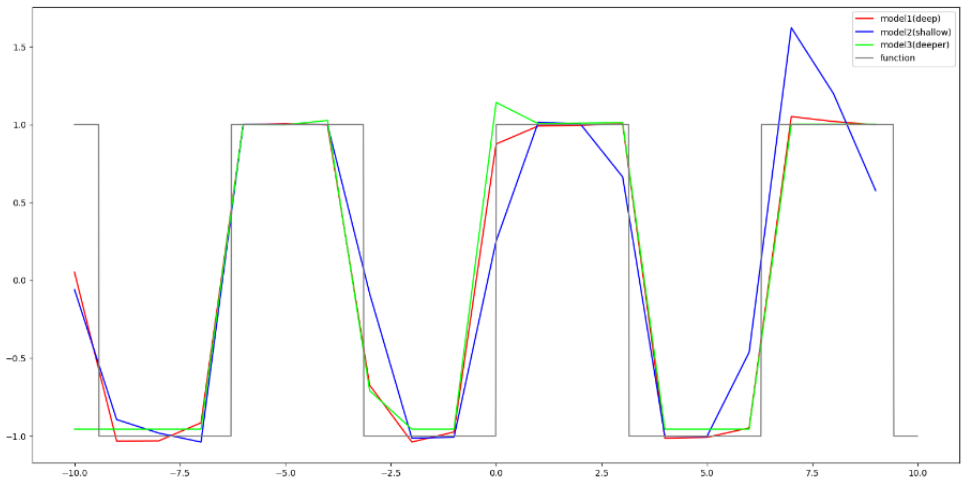
4. Comment on your results. (1%)

我們生出-10 到 10 中間用 f(x) = sin x 生出 100 筆 data，每層的 activation 採用 ReLU(除了最後預測的那層)，optimizer 使用 adam，learning rate 0.01。開始 train 後會發現在一開始深度一點的 NN 的 MSE loss 會下降的較快並在數萬個 epoch 後漸漸趨於穩定。比較三個 model 的 loss 最低點，會發現 model1 是最好的，其次是 model3，最後才是 shallow 的 model2，各自之間有將近十倍的差距。雖然總歸來說，deep 的兩個 Network 確實是筆 shallow 的表現來的好的，但並非越 deep 的表現越好，我們推測是每多一層 layer 就類似在高維空間中摺疊這些 data 的感覺，有時一些較為簡單的 task，硬是疊了太多層的 layer 反而會得到反效果。更極端的情況下甚至會直接"壞掉"(我們用類似的方法疊了 10 層以後，會發現 loss 變得誇張的大，作圖時直接出現一條直線，數萬個 epoch 後仍得不到好的結果)。

BONUS:

Function: f(x) = sgn(sin(x))



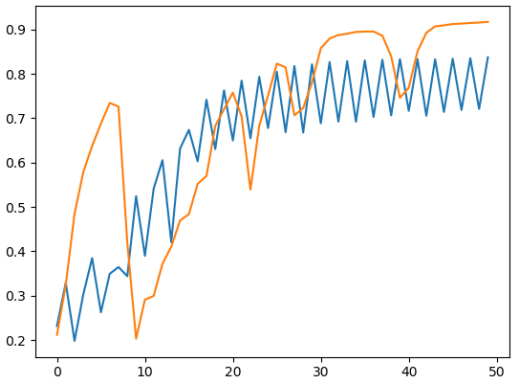


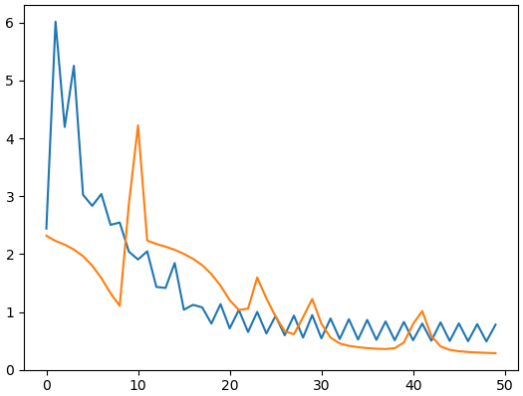
感覺 model3 有點 train 壞了，基本上 model1 還是表現最好的，在這個 task model2 和 model3 之間的最後差異不大，但是一開始 model3(最 deep 的 model)是降得更低的。與上面比較大概可得出雖然 deep 應是較 shallow 好，但疊太多層 layer 卻也不一定更好。

結論：越 deep 不見得越好

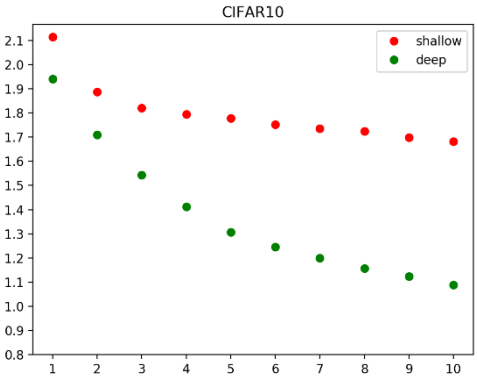
Train on Actual Tasks:

1. Describe the models you use and the task you chose. (0.5%)
   我們用 MNIST 作為第一個 task，使用的 Shallow Model 為一個單層 28 個 Neuron 的 CNN，Deep Model 為一個雙層各 16 個 Neuron 的 CNN





2. In one chart, plot the training loss and the training accuracy of all models. (1%)
   見上面兩張圖
3. Comment on your results. (1%)
   我們使用 optimizer 使用 SGD，learning rate 為 0.001。基本上最後 Deep 的表現會比較好，雖然一開始的震盪跟 shallow 的比較起來算是非常的大，但在 50 個 epoch 後它的 training loss 大概是 shallow 的 $\frac{1}{10}$ (上圖的 y 軸代表 loss 取 log 後的質)。
4. Use more than two models in all previous questions. (bonus 0.25%)



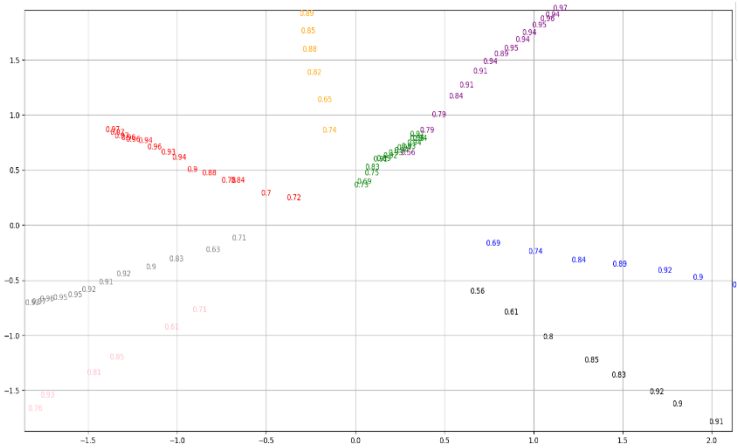
5. Train on more than one task. (bonus 0.25%)
   左圖是我們在 cifar 上用類似的架構時作後得到的 loss 值。

1-2

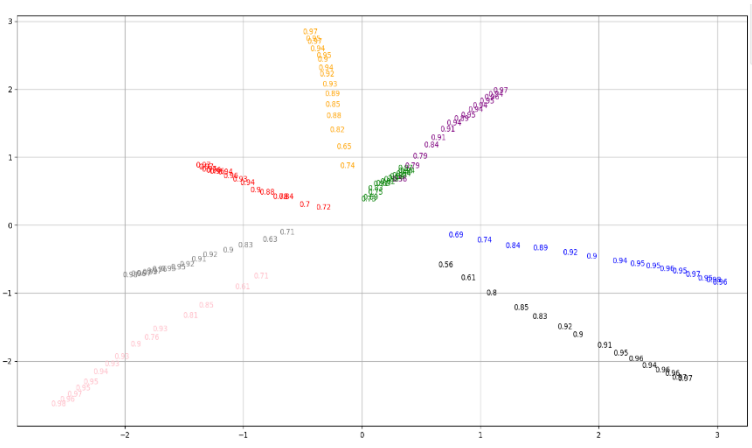
Visualize the optimization process.

1. Describe your experiment settings. (The cycle you record the model parameters, optimizer, dimension reduction method, etc) (1%)
   架構使用兩層各 16 個 Neuron 的 CNN 作 MNIST 的 task，使用 Adam 作為 optimizer，Learning rate 為 0.01，batch_size 15000，並在每次 epoch 結束後記錄一次 model 中的參數，各 train 15 個 epoch，共 8 次，生出八個 model，最後用 PCA 降維法，將所有參數(bias 除外)，降至 2 維。

2. Train the model for 8 times, selecting the parameters of any one layer and whole model and plot them on the figures separately. (1%)



The whole model's params



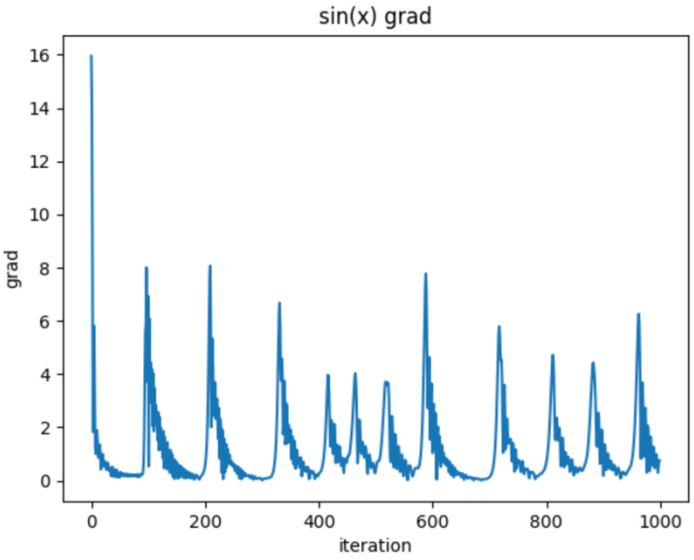
The last layer's params (prediction)

3. Comment on your result. (1%)
   PCA 降維後，會在兩張圖中都看到八個 model 都是以中間做為輻射點，往外走，並越走越小步。Weight 變動的幅度越來越慢，所以越後面的 epoch 移動的越少算是在我們的推測之中，但是每個 model 卻往不同方向前進卻算是出乎我們意料，推測他們可能各自往不同的 Minima 走去了。

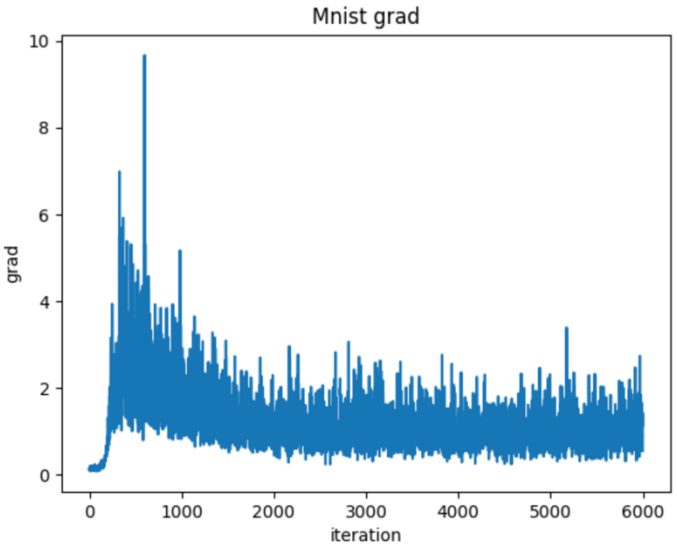
Observe gradient norm during training.

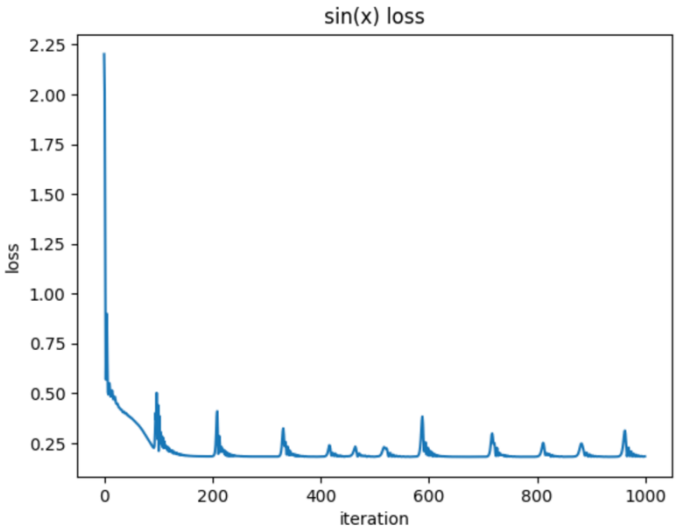
1. Plot one figure which contain gradient norm to iterations and the loss to iterations. (1%)

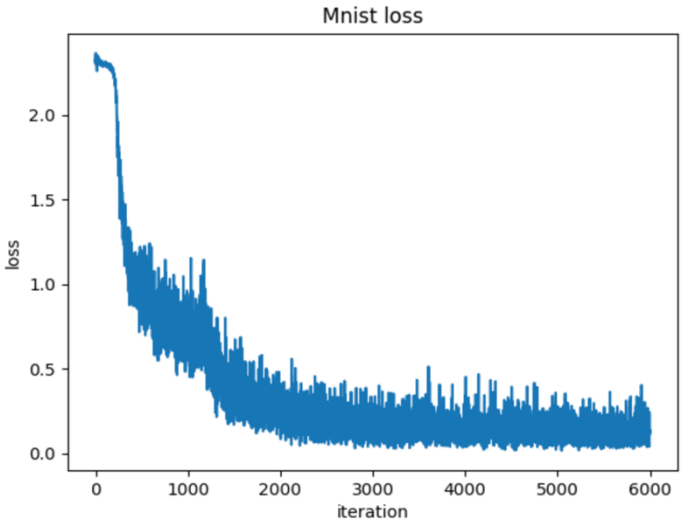
Sin(x)



Mnist







2. Comment your result. (1%)

觀測 sin(x)的 loss 可以看出不需要很多的 epoch 便能很快地降低，因此可以知道越簡單的函式越容易訓練得比較快。

Sin(x)的 model 架構為：

```
class Net(nn.Module):
    def __init__(self,n_feature,n_hidden,n_hid, n_output):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_feature,100)
        self.predict = nn.Linear(100,n_output)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.predict(x)
        return x
```

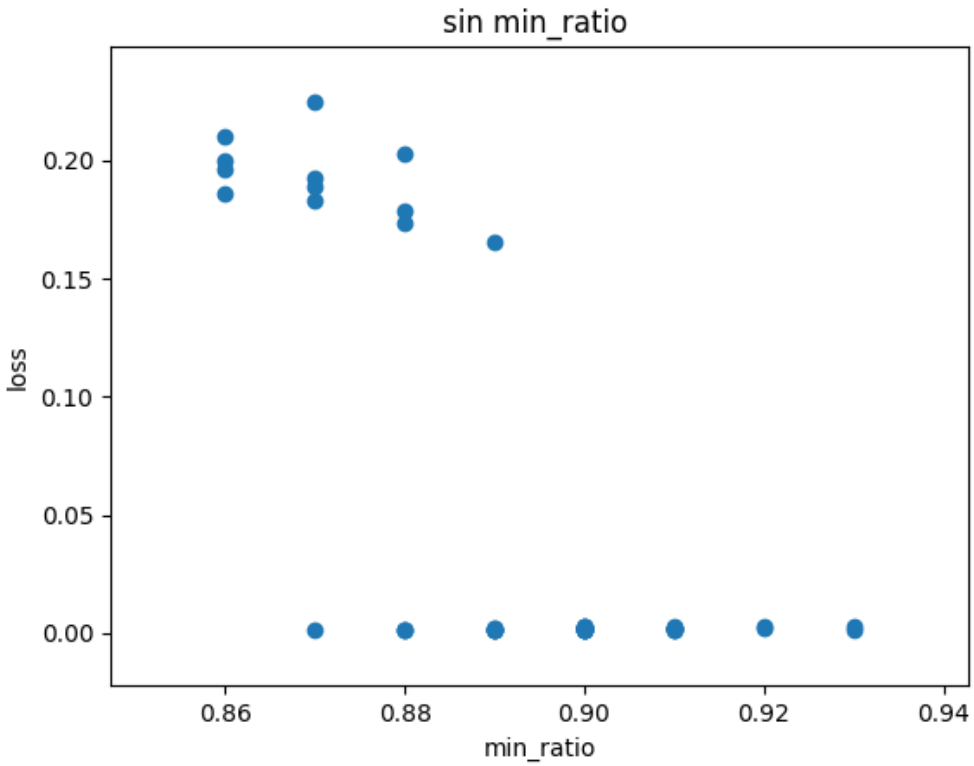
mnist 的 model 架構為：

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 3, 5, 1)
        self.conv2 = nn.Conv2d(3, 5, 5, 1)
        self.fc1 = nn.Linear(4 * 4 * 5, 5)
        self.fc2 = nn.Linear(5, 16)
        self.fc3 = nn.Linear(16, 10)
```

What happens when gradient is almost zero?

1. State how you get the weight which gradient norm is zero and how you define the minimal ratio. (2%)
   我在訓練模型時的每個 epoch 更新最小的 gradient norm，並記錄最小 gradient norm 時的 weight，而 weight 就是 model.parameters 的總和，minimal ratio 是用在 gradient norm 很接近零的地方(最小值)計算他的 hessian 並算他的正 eigenvalue 佔總 eigenvalue 個數的比率

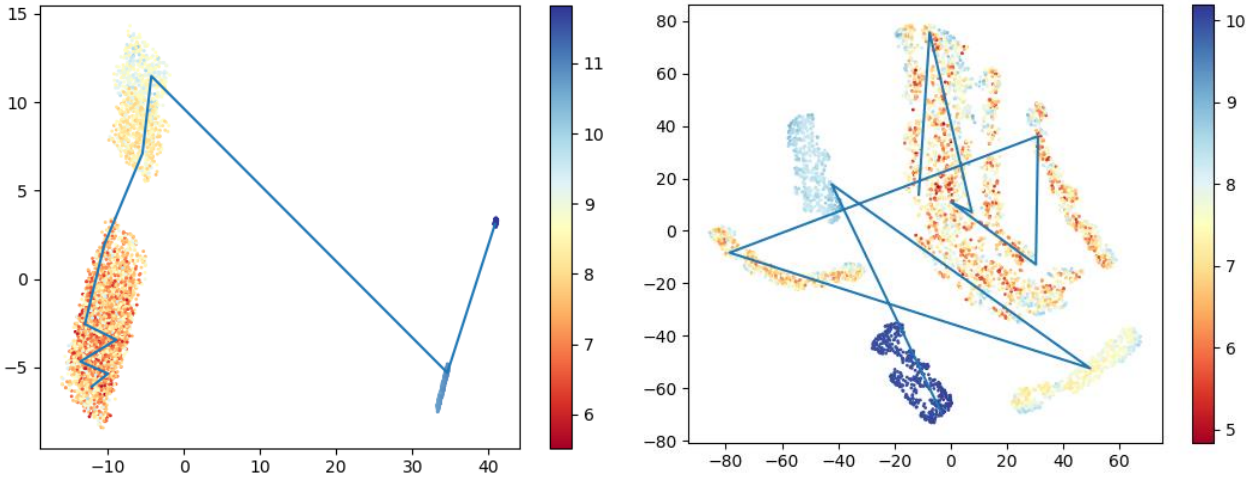
2. Train the model for 100 times. Plot the figure of minimal ratio to the loss. (2%)



3. Comment your result. (1%)
   試過不同的 optimize function 後，可以知道每次 gradient norm 接近零的時候，此點不一定是 local min，有幾次的結果可以觀察出，min ratio 很低，因此可能是在 saddle point 上，在這之後，因為使用牛頓法(上圖)，再經由不同的 initialize 做出比較好的 training 結果後，可以很快地就可以接近想要的 gradient norm 為零的點，因此可以比其他 optimize 的方法更快找到可能是 local min 的點，並可以求出 loss 很低的時候，minimal ratio 很高

Bonus (1%)

1. Use any method to visualize the error surface.



PCA TSNE

2. 我們定義我們的task為去fit $f(x)=x^2$，sample總數共100個。在使用adam作為optimizer(lr=0.01) train過1000 epoch後，我們使用second order optimization：LBFGS，再train 10個epoch，並每次紀錄其weight並隨機在它附近sample 500個點，並利用tsne作了一次圖，發現其走的路徑有點奇怪，決定用PCA降維再做一次，可以看出error surface大致的樣子(右高往左低)，但當gradient降的太快的時候中間那段就sample不出來了。Sample的方法應該跟gradient要更相關才對，但由於時間緊迫，因此沒有嘗試出來。

1-3

Can network fit random labels?

1. Describe your settings of the experiments. (1%)

(a)Task : CIFAR10

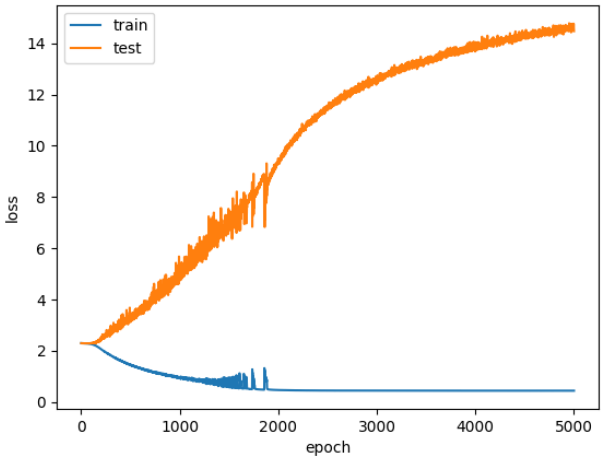
(b)Optimizer : SGD

(c)Learning Rate : 0.01

(d)Batch Size : 256

(e)Architecture(簡介) : 2層CNN+2層DNN

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



Number of parameters v.s. Generalization

1. Describe your settings of the experiments. (1%)

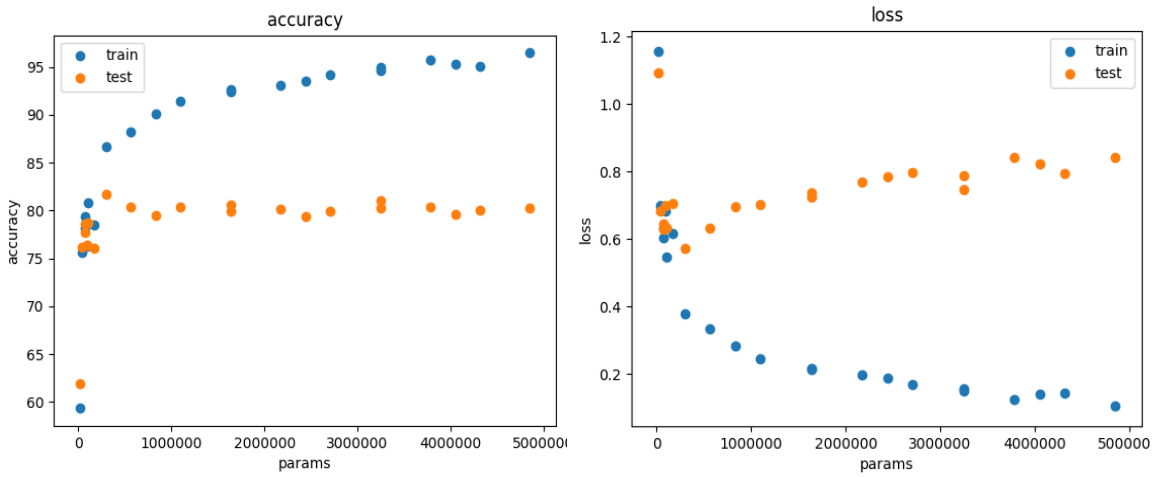
   (a)Task : CIFAR10

   (b)Optimizer : SGD

   (c)Learning Rate : 0.01

   (d)Batch Size : 256

   (e)Architecture : 2 層 CNN+2 層 DNN，並調整 CNN 的 channel 以及 DNN 的 neuron 數來達到 params 的不同。



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

   見上圖

3. Comment your result. (1%)

   參數量從極少開始增加時，會大幅提升 training 與 testing 的準確率，以及大幅降低兩者的 loss，但在參數量達到約 200000 時達到飽和，繼續增加參數量，會持續增加 training 的準確率及降低其 loss，但是 testing 的準確率與 loss 沒有變好，loss 反而提升。因此可知愈多 params 可以更佳 fit training data 但對 testing 產生不穩定的影響。

Flatness v.s. Generalization

Part 1

1. Describe your settings of the experiments. (1%)

   (a)Task : MNIST

   (b)Optimizer : SGD

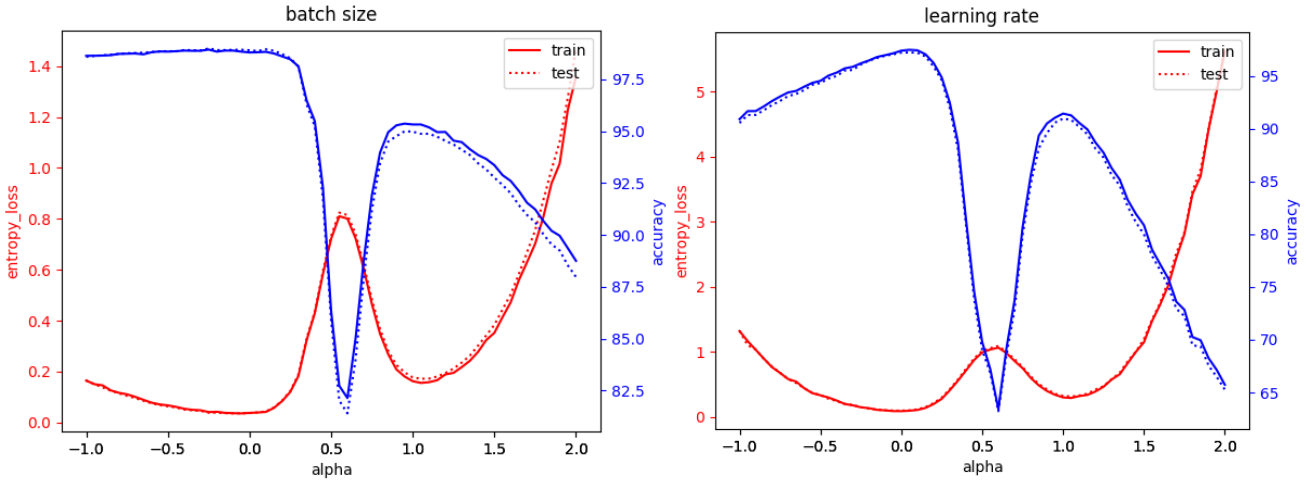
   (c)Learning Rate : 0.01

   (d)Architecture(簡介) : 2 層 CNN+1 層 DNN

   (e)左圖(batch size)：$\theta_1 = 64,\ \theta_2 = 1024$

   右圖(learning rate)：$\theta_1 = 0.01,\ \theta_2 = 0.001$

2. Plot the figures of both training and testing, loss and accuracy to the number of interpolation ratio.



3. Comment your result. (1%)

   可以發現內差係數在 0.5 與 2 時得到最差的結果，可能是因為趨近 0.5 時，結合兩個 model 的 weight，會導致原本 loss 的低谷被另一個 model 影響，因此重疊之後的 model weight 不會使 data set 處於 loss 的最低谷。

Part II

1. Describe your settings of the experiments. (1%)

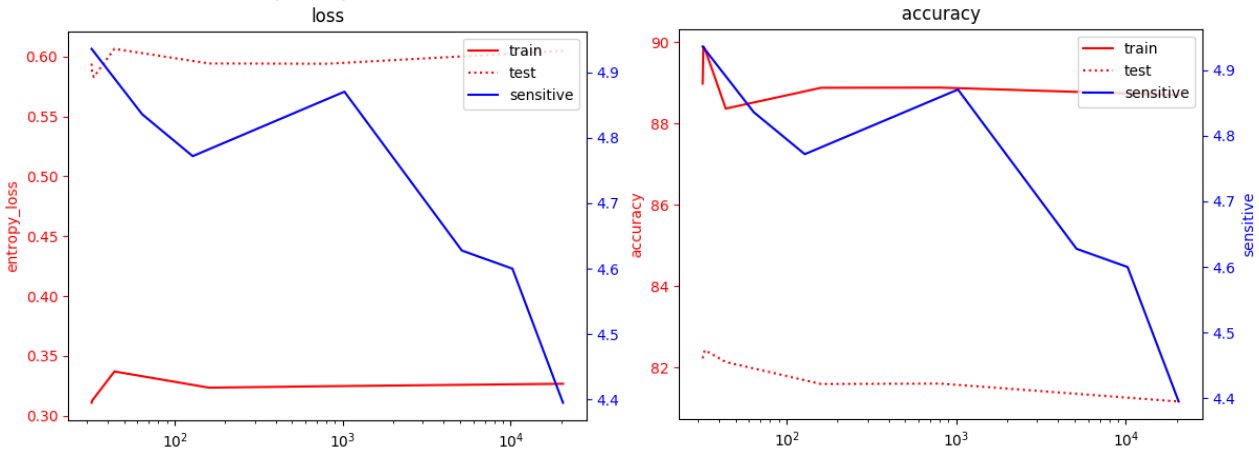
   (a) Task : CIFAR10

   (b) Optimizer : SGD

   (c) Learning Rate : 0.01

   (d) Architecture(簡介) : 2 層 CNN + 2 層 DNN

2. Plot the figures of both training and testing, loss and accuracy, sensitivity to your chosen variable.



3. Comment your Result. (1%)

發現 sensitive 隨著 batch size 的增大而降低，表示 batch size 愈大，loss 谷底愈是平滑。

分工表

楊力權 b03902101 40%
張庭維 b03902093 30%
廖廷浩 b03902102 30%