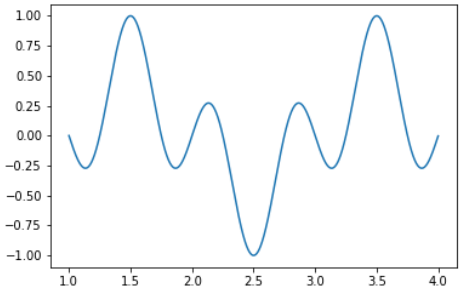
MLDS HW1

R06725007 賴冠廷、R06725015 李尚恩、R06725019 江孟軒

HW1-1

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%)
   Our Function : **cos(2*pi*x) * sin(pi*x)**



皆使用 DNN 的模型，Loss function 為 MSE，optimizer 為 Adam。
從層數多到少依序如下：

```
FuncNet1(
  (fc1): Linear(in_features=1, out_features=5, bias=True)
  (fc2): Linear(in_features=5, out_features=8, bias=True)
  (fc3): Linear(in_features=8, out_features=10, bias=True)
  (fc4): Linear(in_features=10, out_features=10, bias=True)
  (fc5): Linear(in_features=10, out_features=10, bias=True)
  (fc6): Linear(in_features=10, out_features=8, bias=True)
  (fc7): Linear(in_features=8, out_features=5, bias=True)
  (fc8): Linear(in_features=5, out_features=1, bias=True)
)
```

參數量：507

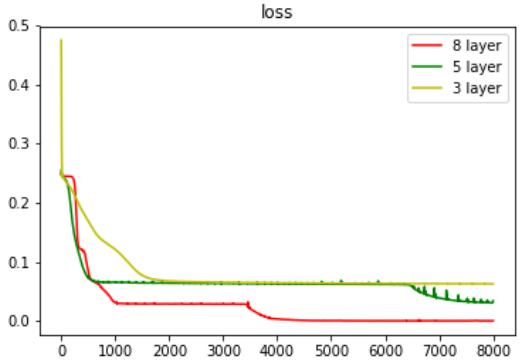
```
FuncNet2(
  (fc1): Linear(in_features=1, out_features=8, bias=True)
  (fc2): Linear(in_features=8, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=14, bias=True)
  (fc4): Linear(in_features=14, out_features=8, bias=True)
  (fc5): Linear(in_features=8, out_features=1, bias=True)
)
```

參數量：504

```
FuncNet3(
  (fc1): Linear(in_features=1, out_features=21, bias=True)
  (fc2): Linear(in_features=21, out_features=20, bias=True)
  (fc3): Linear(in_features=20, out_features=1, bias=True)
)
```

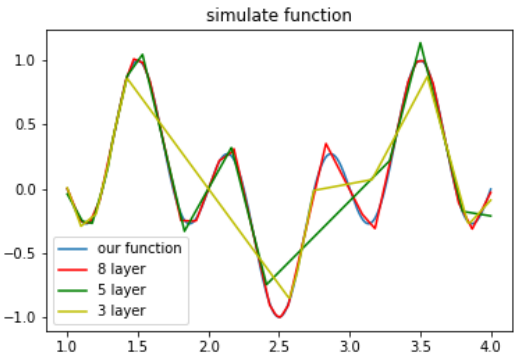
參數量：503

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



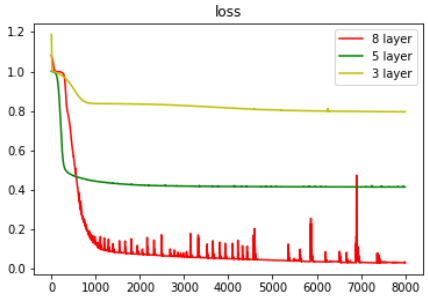
可以發現紅色的線（8 layers）是 loss 下降最快的，大約在 4000 epoch 就趨近於零，綠線（5 layers）一開始也下降很快，但後來就卡在 0.08 左右，而黃線（3 layers）則是一開始就下降得比較慢。

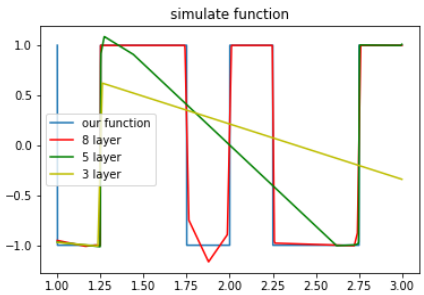
3. In one graph, plot the predicted function curve of all models and the ground-truth function curve. (0.5%)



契合程度為紅線（8 layers）> 綠線（5 layers）> 黃線（3 layers）

4. Comment on your results. (1%)
   越多層的 model 越能更快的 simulate 我們的 function，而且較多層的 model 也能 fit 比較好，第三題的途中可以發現在不同的 model 訓練到收斂後，三層的 model 只能大概抓到上下波動的趨勢，不能精確描述每一個轉折，而八層的 model 能幾乎 fit 整個 function，只有在轉折處有一點小小的偏差。
5. Use more than two models in all previous questions. (bonus 0.25%)
   我們上面使用三種 model 比較
6. Use more than one function. (bonus 0.25%)
   Another Function : **sign(cos(2*pi*x) * sin(pi*x))**





與上面的 function 有差不多的結果，越多層越能 fit

Train on Actual Tasks:

1. Describe the models you use and the task you chose. (0.5%)我們選擇 MNIST 的資料集，並使用三種不同 hidden layers 及不同 dimension 的 CNN，loss function 為 cross entropy，optimizer 為 Adam。我們 model 架構如下：

Layer (type)	Output Shape	Param #
conv2d_217 (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d_166 (MaxPoolin	(None, 13, 13, 8)	0
dropout_226 (Dropout)	(None, 13, 13, 8)	0
flatten_109 (Flatten)	(None, 1352)	0
dense_228 (Dense)	(None, 32)	43296
dropout_227 (Dropout)	(None, 32)	0
dense_229 (Dense)	(None, 10)	330

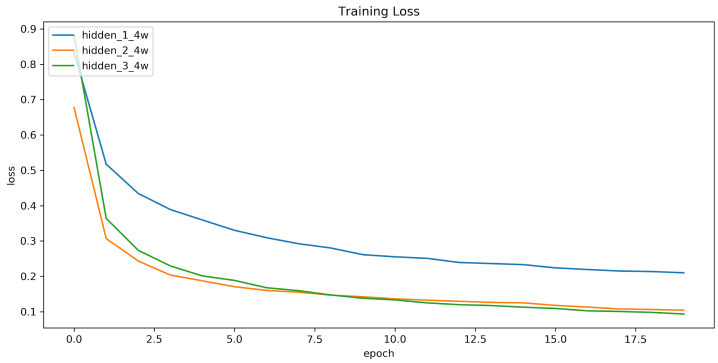
Total params: 43,706
Trainable params: 43,706
Non-trainable params: 0

Layer (type)	Output Shape	Param #
conv2d_218 (Conv2D)	(None, 26, 26, 38)	380
max_pooling2d_167 (MaxPoolin	(None, 13, 13, 38)	0
conv2d_219 (Conv2D)	(None, 11, 11, 38)	13034
max_pooling2d_168 (MaxPoolin	(None, 5, 5, 38)	0
dropout_228 (Dropout)	(None, 5, 5, 38)	0
flatten_110 (Flatten)	(None, 950)	0
dense_230 (Dense)	(None, 32)	30432
dropout_229 (Dropout)	(None, 32)	0
dense_231 (Dense)	(None, 10)	330

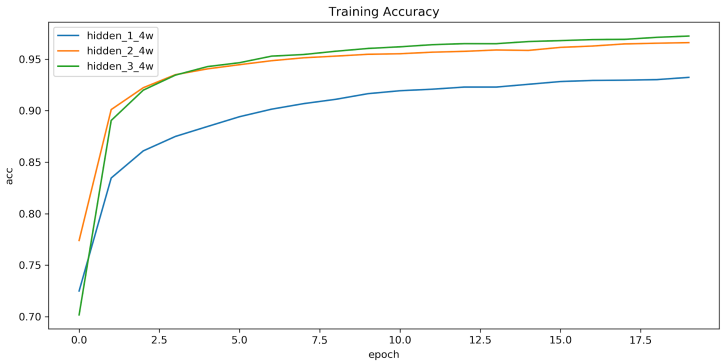
Total params: 44,176
Trainable params: 44,176
Non-trainable params: 0

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 26, 26, 48)        480
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 13, 13, 48)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 11, 11, 48)        20784
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 5, 5, 48)          0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 3, 3, 48)          20784
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 1, 1, 48)          0
_________________________________________________________________
dropout_19 (Dropout)         (None, 1, 1, 48)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 48)                0
_________________________________________________________________
dense_19 (Dense)             (None, 32)                1568
_________________________________________________________________
dropout_20 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_20 (Dense)             (None, 10)                330
=================================================================
Total params: 43,946
Trainable params: 43,946
Non-trainable params: 0
```

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



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



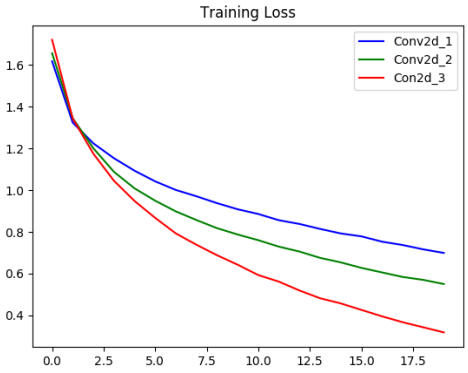
4. Comment on your results. (1%)
   由上面兩張圖可以清楚得知，有兩層 hidden layers 的 model 其成效皆遠遠好於只有一層的 model，而兩層 hidden layers 與三層的表現則是差不多，三層的表現有略好一點。因此我們可以知道同樣參數下加深 model 有助於 training 的表現。
5. Use more than two models in all previous questions. (bonus 0.25%)
   如上述
6. Train on more than one task. (bonus 0.25%)
   我們另外也使用了 CIFAR10 做測試，model 的參數如上述，也是分別比較了一層， 兩層， 與三層的 CNN，總參數量皆為 12 萬左右，model 架構如下:

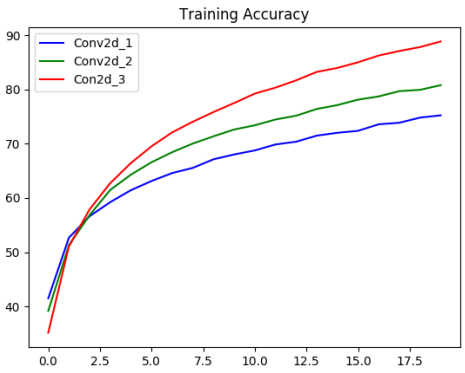
```
Model_1(
  (conv1): Conv2d (3, 8, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(3, 3), dilation=(1, 1))
  (fc1): Linear(in_features=1800, out_features=64)
  (fc2): Linear(in_features=64, out_features=64)
  (fc3): Linear(in_features=64, out_features=10)
)
```

```
Model_2(
  (conv1): Conv2d (3, 10, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d (10, 48, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (fc1): Linear(in_features=1728, out_features=64)
  (fc2): Linear(in_features=64, out_features=64)
  (fc3): Linear(in_features=64, out_features=10)
)
```

```
Model_3(
  (conv1): Conv2d (3, 15, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d (15, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (fc1): Linear(in_features=512, out_features=64)
  (fc2): Linear(in_features=64, out_features=64)
  (fc3): Linear(in_features=64, out_features=10)
)
```

另外從下圖一樣也可以看到在同樣的參數量下，多層的 CNN 在 loss 與 accuracy 上都有較佳的表現。

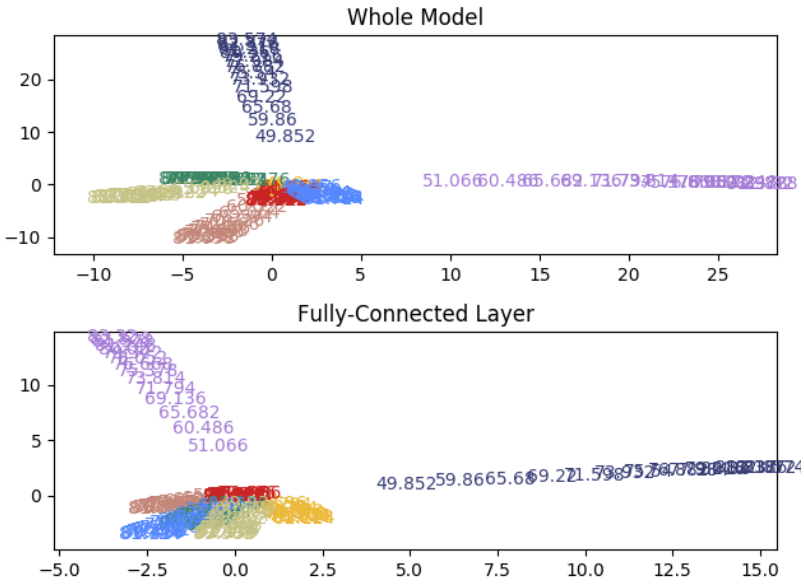




HW1-2

Visualize the optimization process

1. Describe your experiment settings. (The cycle you record the model parameters, optimizer, dimension reduction method, etc) (1%)
   我們使用了兩層的 CNN train 在 CIFAR10 的資料集上，optimizer 為 Adam。每次 training 皆 train 45 個 epoch 並且每 3 個 epoch 就記錄一次參數(weights) ，最後使用 PCA 做降維。
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%)
   下圖為 whole model 與 Fully-Connected Layer 降維成 2 維的結果圖：

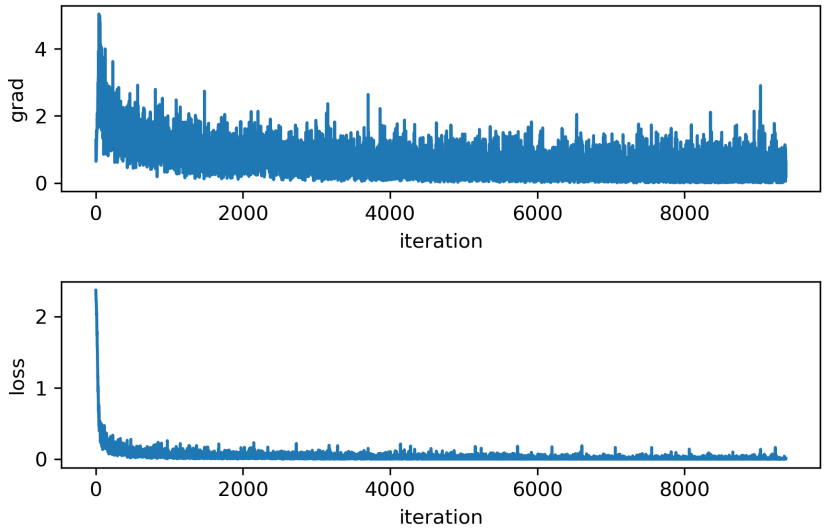


3. Comment on your result. (1%)
   可以觀察出雖然每一次的 training event 初始的位置都一樣，但是參數更新的方向皆不同，而且從資料點(Accuracy)的疏密度可以看出一開始的 gradient 較大，參數移動的位置比較多。

Observe gradient norm during training

1. Plot one figure which contain gradient norm to iterations and the loss to iterations. (1%)
   此題的 model 架構相同，然而改為 train 在 MNIST 的資料集上，總共 train 了 20 個 epoch(9380

個 iterations)，結果如下圖。



2. Comment your result. (1%)
   我們發現即便 loss 已經沒有多大的震盪，幾乎已經趨近於 0，但 gradient norm 還是在一個區間內不斷震盪，並不等於 0，所以我們可以推論或許 model 還沒有走到一個 critical point。

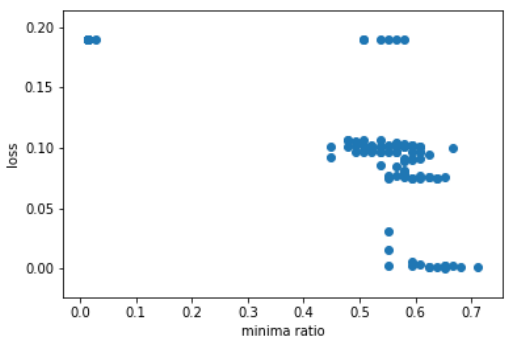
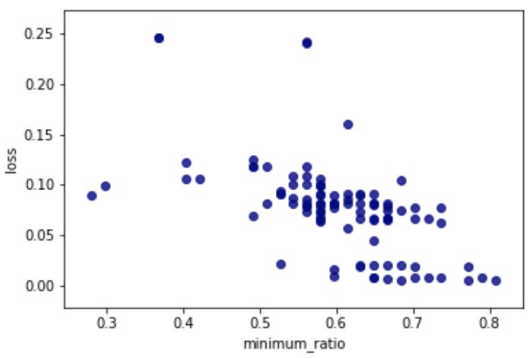
What happens when gradient is almost zero

1. State how you get the weight which gradient norm is zero and how you define the minimal ratio. (2%)
   Our Function : **cos(2*pi*x) * sin(pi*x)**
   我們設定 train 20000 epochs，並先使用 MSE 作為 Loss function，當到達 16000 epochs 時將 objective function 改成 gradient norm，並使用它來 update 參數，當當次的 gradient norm < 0.005 或是 train 到 20000 epochs 結束時，算 hessian matrix。
   從 hessian matrix 去算 eigenvalues 數量，並且定義 minimal ratio 為所有 eigenvalues 數量分之正的 eigenvalues 數量。

2. Train the model for 100 times. Plot the figure of minimal ratio to the loss. (2%)
   我們使用了兩種不同 model 結構來看結果：

	model1	model2

model structure	`FuncNet1(`
	`  (fc1): Linear(in_features=1, out_features=6, bias=True)`
	`  (fc2): Linear(in_features=6, out_features=5, bias=True)`
	`  (fc3): Linear(in_features=5, out_features=3, bias=True)`
	`  (fc4): Linear(in_features=3, out_features=1, bias=True)`
	`)`
		`FuncNet1(`
		`  (fc1): Linear(in_features=1, out_features=4)`
		`  (fc2): Linear(in_features=4, out_features=8)`
		`  (fc3): Linear(in_features=8, out_features=1)`
		`)`

loss and minimal ratio		

3. Comment your result. (1%)
   共通點：
   兩張圖都可以看出有一層一層的關係，並且越低的 loss 有 minimal ratio 越高的趨勢，因此可以驗證當 loss 很接近 0 時有比較高的機率是走到 local minimum，並且許多點會集中在特定的 loss，而其算出來的 minimal ratio 也相對比較小，表示最後停在 saddle point 的機率較高。
   model 差異：
   在 model1 中 loss 集中在三種 level，當 loss=0.19 時藉由 minimal ratio 可以得知應該有一個 local maximum(ratio 接近 0，表示 eigenvalues 幾乎都是負的)以及 saddle point，loss 在 0.1 附近也有一個 saddle point(minimal ratio 接近 0.5)；在 model2 的部分相對 model1 在停留 loss 高的次數較少，可以推測其 error surface 在 loss 較高處的面積相對小，而結果一部分分布在 loss 接近 0.1 的 saddle point 以及 loss 靠近 0 的 local minimum。

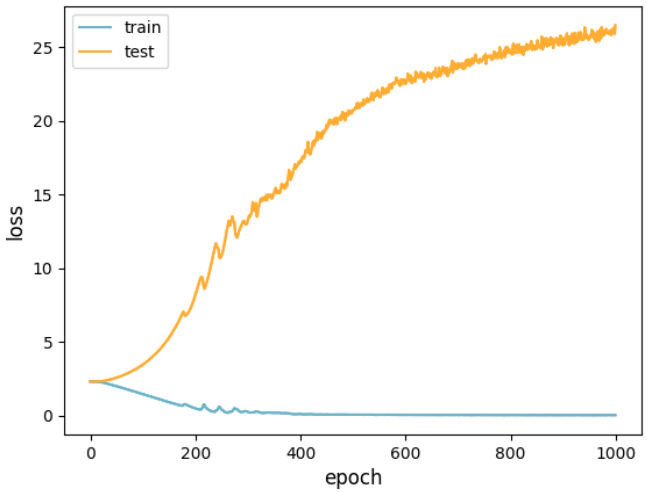
HW1-3

Can network fit random variables

1. Describe your experiment settings(1%)
   我們使用了三層的 CNN 並 train 在 CIFAR10 上，optimizer 為 Adam，learning rate 為 0.00005，train 1000 個 epoch。以下為 model structure:

```
Conv_3(
  (conv1): Conv2d (3, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d (32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (fc1): Linear(in_features=512, out_features=256)
  (fc2): Linear(in_features=256, out_features=256)
  (fc3): Linear(in_features=256, out_features=10)
)
```

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



   以上為打亂 label 後 training 跟 testing 對 epoch 關係圖。可以觀察到 CNN 是有辦法把五萬筆的 training data 全部背起來，其 loss 接近 0。在訓練過程中我們發現原本的 learning rate (0.001)對於打亂順序後太大了，導致一開始訓練時 loss 一直下不去，我們猜測是其 error surface 充滿很多狹小的山谷，導致參數更新幅度太大就卡住下不去，將 learing rate 調整後(0.001 -> 0.00005)就可以 train 下去了。

Number of parameters v.s. Generalization

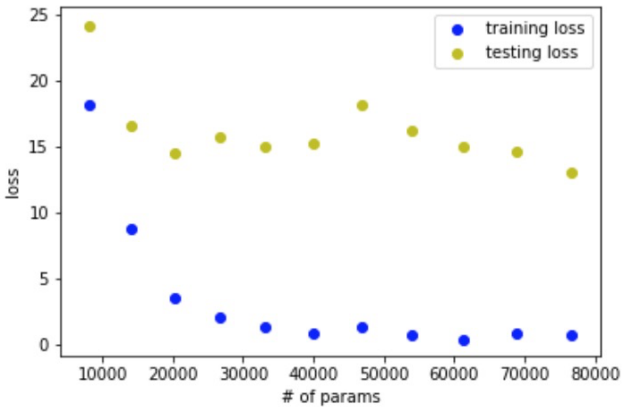
1. Describe your experiment settings(1%)
   我們 train 在 MNIST 上，使用的 DNN 結構如下：

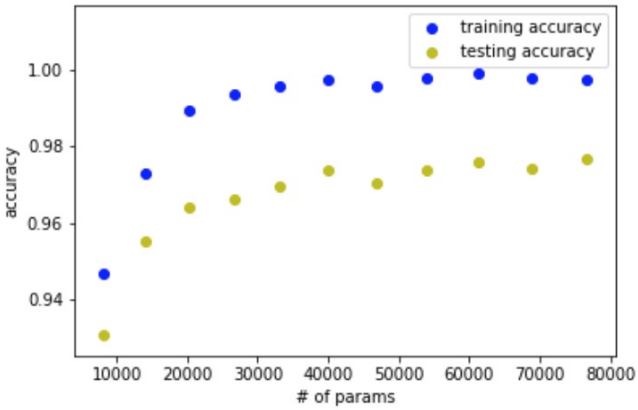
```
Net(
  (fc1): Linear(in_features=784, out_features=10, bias=True)
  (fc2): Linear(in_features=10, out_features=10, bias=True)
  (fc3): Linear(in_features=10, out_features=10, bias=True)
  (fc4): Linear(in_features=10, out_features=10, bias=True)
  (relu): ReLU()
)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=17, bias=True)
  (fc2): Linear(in_features=17, out_features=17, bias=True)
  (fc3): Linear(in_features=17, out_features=17, bias=True)
  (fc4): Linear(in_features=17, out_features=10, bias=True)
  (relu): ReLU()
)
```

   兩層的 hidden layer，而控制參數變化就是改變 hidden layer 中的神經元數量
   分別使用：10,17,24,31,38,45,52,59,66,73,80 個神經元，共 11 種不同參數的 model，並 train 35 個 epoch，此時 loss 已經下降的很緩慢。

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





3. Comment your result (1%)
   相同的神經網路結構中，越多的參數能夠 fit 這個 dataset，而當越多(20000-30000)的參數 training loss / accuracy 仍然有在下降，但 testing loss / accuracy 就幾乎定型了，可以知道當 model 已經能夠 fit 再增加參數也不會導致 overfitting。

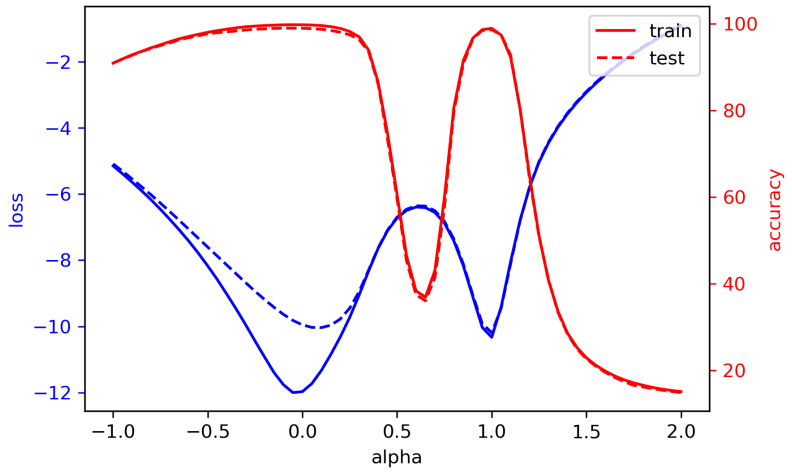
Flatness v.s Generalization

Part I

1. Describe your experiment settings(0.5%)
   我們使用了兩層的 CNN 並 train 在 MNIST 上，optimizer 為 Adam，learning rate 為 0.001， train 20 個 epoch，並使用 batch size 64 與 1024。以下為 model structure:

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=500, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=10, bias=True)
)
```

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



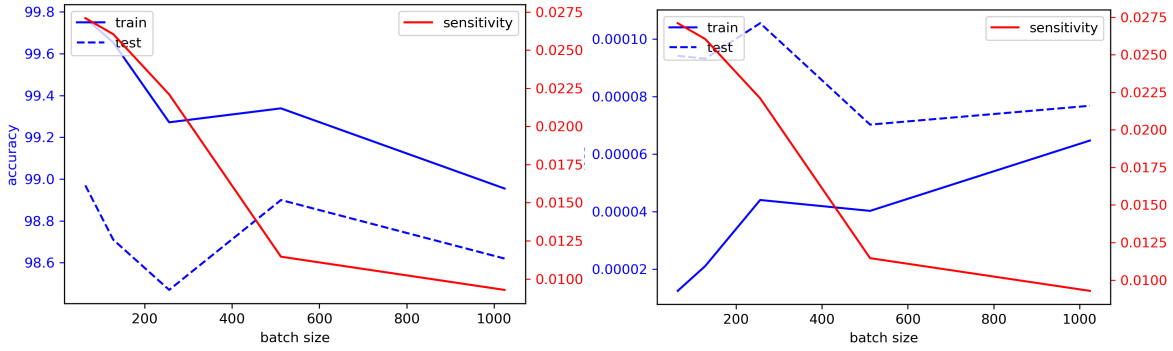
3. Comment your result (1%)
   由上圖我們可以發現在 alpha 等於 0.0 或 1.0 時，也就是不論在較大的 batch size 還是較小的 batch size，都會走到一個類似 local minima 的地方，而其中在這個切面上較小的 batch size 的附近較平坦，較大的 batch size 則較陡峭，這現象有可能意味著較小的 batch size 的 generalization 能力較好。

Part II

1. Describe your experiment settings(0.5%)
   我們使用了兩層的 CNN 並 train 在 MNIST 上，optimizer 為 Adam，learning rate 為 0.001， train 20 個 epoch，並使用 batch size 64、128、256、512 與 1024。以下為 model structure:

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=500, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=10, bias=True)
)
```

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



3. Comment your result (1%)
   由上圖我們可以發現，首先 batch size 越大的 model 其 loss 越高 accuracy 越低。然而 batch size 越大的 model 其 sensitivity 越小，而相反地 batch size 越小的 model 其 sensitivity 則越大。因此若根據 sensitivity 推測，batch size 越大的 model 其 generalization 的能力可能是越好的。