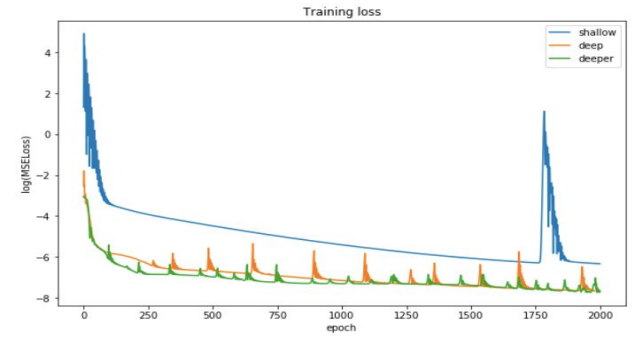
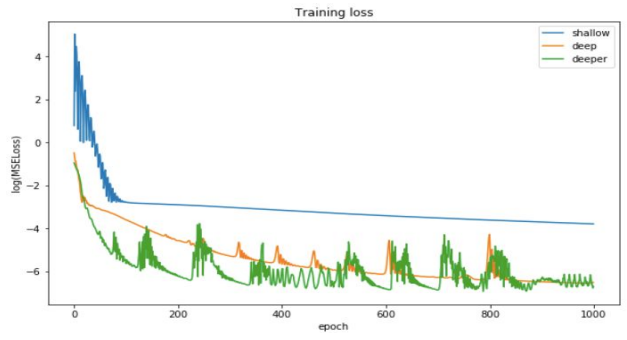
MLDS HW1

HW1-1(1)Simulate a Function

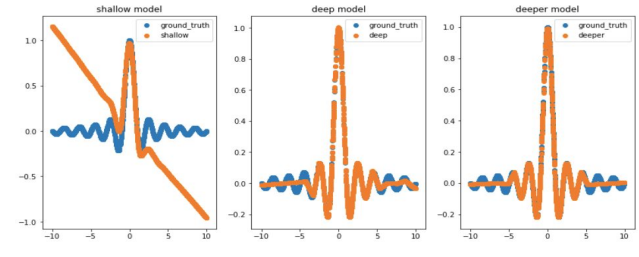
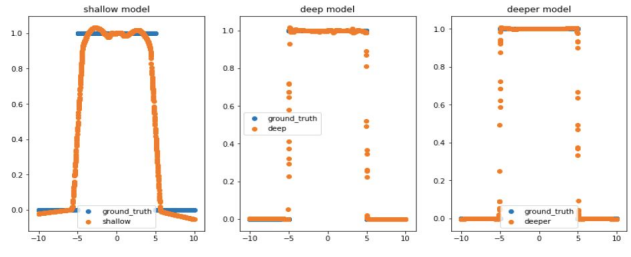
1. Describe the models you use, including the number of parameters (at least two models) and the function you use. (0.5%)
   * 我們實作兩種函數，套用在三種不同深度參數量相近的network， 並依層數命名為Shallow、Deep及Deeper。Activation function皆為relu。Loss function皆為MSE loss。Optimizer皆使用Adam(learning rate= 5e-3)，Training epoch= 2000。每個epoch會做一次loss的紀錄。
   * 使用兩種函數做測試：y= sinc(x) 、y= rect(x)，當x的絕對值小於等於5時輸出為1，其餘則輸出為0。訓練資料為在x= [-10 10]區間中隨機取樣出2000點。

Model 架構	參數量
Shallow((fc1): Linear(in_features=1, out_features=1880, bias=True) (fc2): Linear(in_features=1880, out_features=1, bias=True)) Deep((fc1): Linear(in_features=1, out_features=30, bias=True) (fc2): Linear(in_features=30, out_features=60, bias=True) (fc3): Linear(in_features=60, out_features=60, bias=True) (fc4): Linear(in_features=60, out_features=1, bias=True)) Deeper((fc1): Linear(in_features=1, out_features=14, bias=True) (fc2): Linear(in_features=14, out_features=20, bias=True) (fc3): Linear(in_features=20, out_features=30, bias=True) (fc4): Linear(in_features=30, out_features=30, bias=True) (fc5): Linear(in_features=30, out_features=30, bias=True) (fc6): Linear(in_features=30, out_features=30, bias=True) (fc7): Linear(in_features=30, out_features=30, bias=True) (fc8): Linear(in_features=30, out_features=30, bias=True) (fc9): Linear(in_features=30, out_features=1, bias=True))	parameters in shallow model: 5641 parameters in deep model: 5641 parameters in deeper model: 5639

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

y= sinc(x)	y= rect(x)
	

3. In one graph, plot the predicted function curve of all models and the ground-truth function curve. (0.5%)
   第1801個epoch時的預測圖形。

y= sinc(x)	y= rect(x)
	

4. Comment on your results. (1%)

a. 觀察：

＊Training loss

深層network的收斂速度遠比shallow network快，其中疊更多層的deeper收斂速度又更勝於deep，即使最後訓練出來的loss其實相差不遠。此外，在收斂的過程中皆可以觀察到ripple的現象。

＊與ground truth比較

深層network的表現顯然較shallow network佳。在training的過程中，我們觀察到預測值很快就能在x=0附近貼近真實值，再隨著epoch數增加慢慢向兩側（遠離x=0處）縮小與真實值的差距。

b. 討論：

在同樣的參數量下，無論在收斂速度與最後的loss上，較深的network皆比shallow network表現得佳，這可能是由於深層的架構更能夠產生較大的function set，而funtion set較小的shallow network無論怎麼在loss hyperplane上走，都很難走到更低之處。此外，之所以deep與deeper的表現看不太出差異，可能是由於測試函數沒有複雜到其中一個model無法學出來的情況，使得收斂情形與接近實際值的狀態皆不錯。另外，我們觀察到的Loss ripple，可能是參數隨著梯度方向移動的時候，在hyperplane上誤踩到了loss更高的地方，隨即使梯度突增造成loss的震盪。

5. Use more than two models in all previous questions. (bonus 0.25%)
   Done.
6. Use more than one function. (bonus 0.25%)
   Done.

HW1-1(2)Train on Actual Tasks

1. Describe the models you use and the task you chose. (0.5%)
   我們實作兩種task，各套用在三種不同深度參數量相近的network上，並依層數命名為Shallow、Deep及Deeper。Activation function皆為relu，部分層間有做maxpooling，最後以一層output layer作結。Loss function皆為Cross entropy。Optimizer皆為Adam（learning rate= (MNIST)5e-4/ (CIFAR)1e-3）。Training epoch皆為50，Batch size= (MNIST)256/ (CIFAR)192。每個batch會做一次accuracy與loss的紀錄。

＊MNIST

Model 架構	參數量
CNN_shallow((conv1): Sequential((0): Conv2d(1, 66, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3)) (1): ReLU() (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)) (fc1): Linear(in_features=12936, out_features=10, bias=True)) CNN_deep((conv1): Sequential((0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)) (1): ReLU() (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False) (3): Conv2d(32, 48, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)) (4): ReLU() (5): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False) (6): Conv2d(48, 72, kernel_size=(5, 5), stride=(2, 2), padding=(1, 1)) (7): ReLU()) (fc1): Linear(in_features=648, out_features=10, bias=True)) CNN_deeper((conv1): Sequential((0): Conv2d(1, 20, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)) (1): ReLU() (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False) (3): Conv2d(20, 36, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)) (4): ReLU() (5): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False) (6): Conv2d(36, 62, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1)) (7): ReLU() (8): Conv2d(62, 74, kernel_size=(3, 3), stride=(1, 1)) (9): ReLU() (10): Conv2d(74, 102, kernel_size=(1, 1), stride=(1, 1)) (11): ReLU()) (fc1): Linear(in_features=918, out_features=10, bias=True))	parameters in shallow model: 132670 parameters in deep model: 132242 parameters in deeper model: 132624

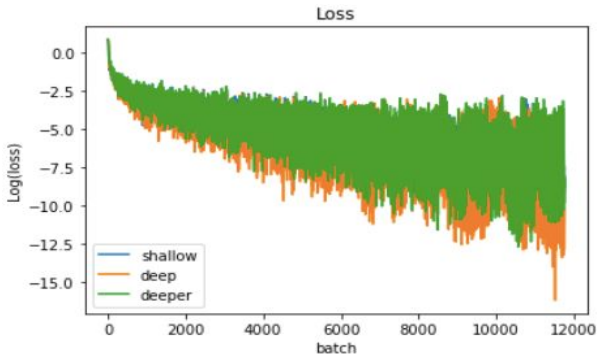
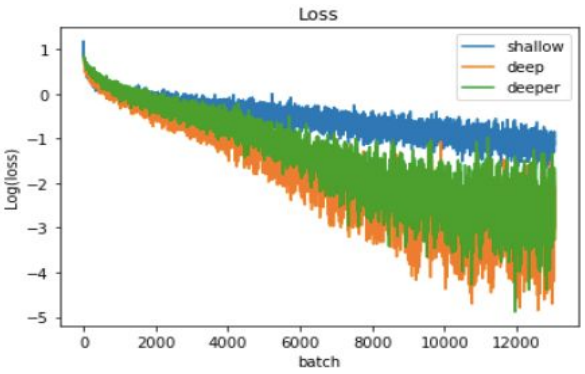
＊CIFAR

Model 架構	參數量

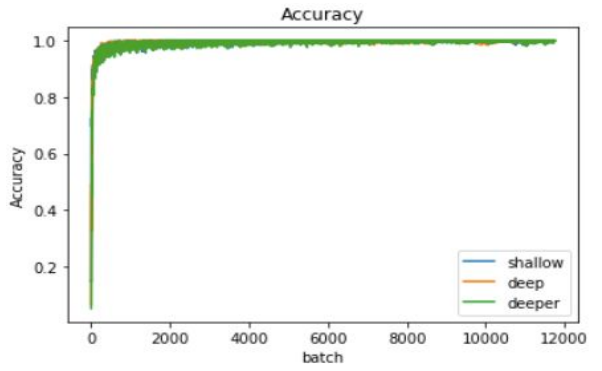
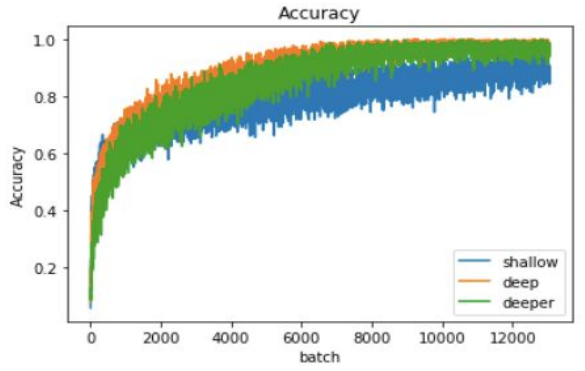
```
ShallowNet(
  (conv1): Sequential(
    (0): Conv2d(3, 51, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  )
  (fc1): Linear(in_features=13056, out_features=10, bias=True)
)
DeepNet(
  (conv1): Sequential(
    (0): Conv2d(3, 22, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (3): Conv2d(22, 35, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (6): Conv2d(35, 75, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (7): ReLU()
  )
  (fc1): Linear(in_features=4800, out_features=10, bias=True)
)
DeeperNet(
  (conv1): Sequential(
    (0): Conv2d(3, 20, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (3): Conv2d(20, 25, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (6): Conv2d(25, 31, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
    (9): Conv2d(31, 46, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (10): ReLU()
    (11): Conv2d(46, 50, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (12): ReLU()
  )
  (fc1): Linear(in_features=800, out_features=10, bias=True)
)
```

```
parameters in shallow model:
134446
parameters in deep model:
134667
parameters in deeper model:
134707
```

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

MNIST	CIFAR
	

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

MNIST	CIFAR
	

4. Comment on your results. (1%)
   a. 觀察：

      * MNIST

      無論使用何種model，其loss的收斂速度都很快，但當epoch數持續增加時，除了因為數量級過小使得震盪看起來很大，也能會看到deep network的loss能夠比deeper更低。在accuracy的結果中，三種network準確率很快就能達到高原值。

      * CIFAR

在loss結果中，shallow的收斂速度明顯較其他兩者慢，隨著epoch數增加deep network的loss也可以比deeper更低。accuracy的結果顯示shallow的表現與其他network有落差，deep與deeper除了在初期的收斂速度有些差異外，最後收斂得出的表現很相近。

b. 討論：
相較於CIFAR，MNIST相對較為簡單、雜訊也較小，因此推論其收斂的速度會較為迅速，短短幾個epoch就能得到很好的結果，所以不容易看出深淺模型在訓練表現上的差異。CIFAR的情況中，我們能觀察到deep的表現似乎較佳，這或許會與「深比較好」的認知有些差異，但事實上，「好」是在於更深的模型代表的是更廣的function set，使得它更有機會包含loss hyperplane中的local minima。這次結果也許是training data中的雜訊、tune的參數以及network的設計導致deep收斂相對較快，但其實也可以從結果看到若epoch數更增加後，其實兩者的表現是差不多的。

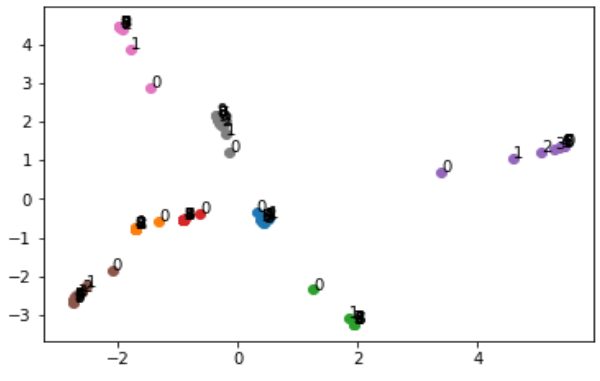
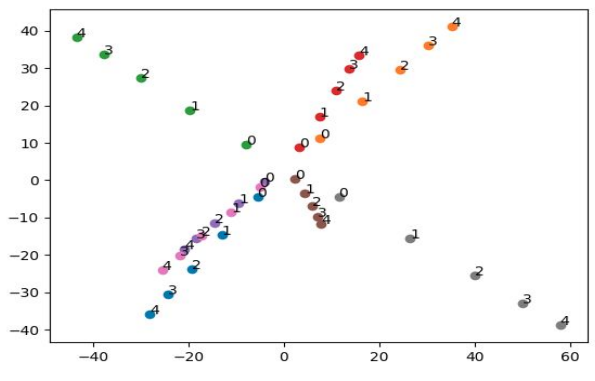
5. Use more than two models in all previous questions. (bonus 0.25%)
Done.
6. Train on more than one task. (bonus 0.25%)
Done.

HW1-2(1)Visualize the optimization process

1. Describe your experiment settings. (1%)
我們使用自定義函數(sinc) 和mnist、cifar-10來訓練，並觀察神經網路最佳化的過程，用同樣的網路架構重複八次訓練，並記錄下每一次網路內的權重。以下是選用的參數：

項目	Sinc function	MNIST
Epoch	7000	10
Learning rate	5e-3	5e-3
Optimizer	Adam	Adam
Loss	MSE	Cross Entropy
Dimension reduction method	PCA	PCA

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%)

Sinc function	MNIST
	

3. Comment on your result. (1%)
可以觀察到在降維後的空間中，初始的權重大多集中在中間區域，在Pytorch的框架中全連接層的參

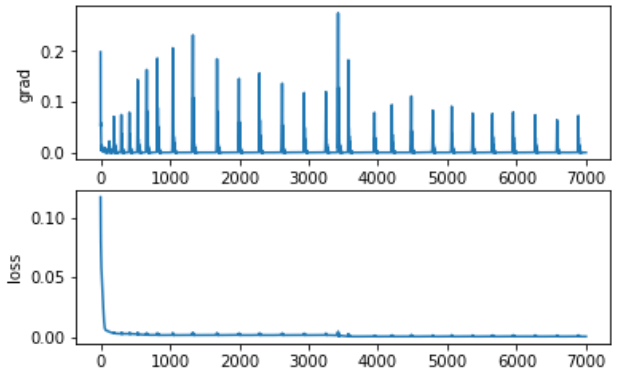
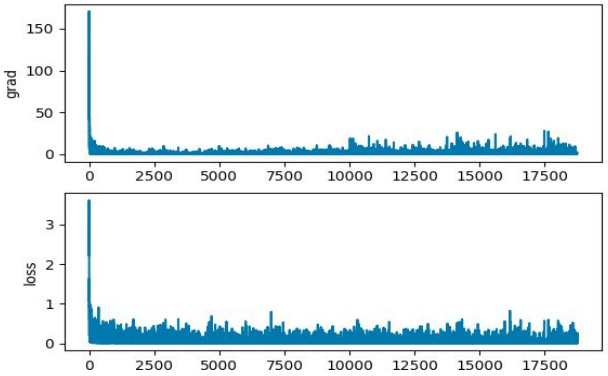
數初始化預設是採用Xavier initialization，我們推論每一次訓練用Xavier初始化的參數大概就是落在中間區域。而隨著訓練，神經網路在降維空間向外尋找讓loss更低的權重，有趣的是每一次訓練得到的參數是截然不同的。

```
def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)
```

Pytorch中Linear class初始化參數的function

HW1-2(2)Observe gradient norm during training.

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

Sinc function	MNIST
	

2. Comment your result. (1%)
我們觀察訓練過程中神經網路的梯度範數(gradient norm)，上圖為用loss finction背向傳播來訓練的過程，gradient norm很快就隨著loss的減少降到接近0，不過可以看到過程中不時會有gradient norm暴增的情況出現。而mnist的情況較為混亂，因為使用的是minibatch的方式訓練，記錄每一個batch訓練後的梯度及loss，比自定義函數的情況更富有隨機性。

HW1-2(3)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%)
這個部分先將要觀察的神經網路先訓練好，接著將背向傳播(back propagation) 的來源從損失函數(loss function) 換成梯度範數(gradient norm)，讓訓練過程的目標轉成「降低梯度」，再繼續訓練。第二次訓練完之後，計算神經網路的Hessian matrix，接著將Hessian matrix的特徵值也計算出來，並用 $\frac{\text{大於0的特徵值數量}}{\text{所有特徵值的數量}}$ 的數量作為minimal ratio。

定義”calculate_grad2()”去計算model裡的grad，這個函式內是用aurograd.grad()去計算每個權重的梯度，是PyTorch在v0.2.0版中更新的功能，主要的差異是：這個梯度是包含在計算圖之中的，也就是他可以用backward()計算梯度，因此可以計算「梯度的梯度」，降低梯度的訓練便是靠grad_norm.backward()來實現的。
”calculate_hessian3()”則是用來計算hessian矩陣的函式，透過呼叫兩次autograd.grad()來計算二階梯度，這裡比較麻煩的是autograd.grad(f, x)中f(被微分的變數) 必須是一個純量，而x可以是一個向量，所以從loss算出grad之後必須用迴圈的方式分別對每個grad中的值算二次微分，最後再把值concat進一個預先宣告的tensor中。

至於"model.parameters()"這個method，給出來的權重並不是一個一維向量，而是跟整個神經網路形狀有關，所以為了要符合autograd.grad()的一些使用限制，只好也用for迴圈，遞迴所有的權重，十分麻煩。而"minimal_ratio()"則是可以計算出矩陣特徵值大於0的比例。

```
def calculate_grad2(model, loss):
    grad_norm = 0
    grad_params = torch.autograd.grad(loss, model.parameters(), create_graph=True,
    for grad in grad_params:
        grad_norm += grad.pow(2).sum()
    return grad_norm

def minimal_ratio(matrix):
    w, _ = np.linalg.eig(matrix)
    return (w > 0).mean()
```

```
def calculate_grad2(model, loss):
    grad_norm = 0
    grad_params = torch.autograd.grad(loss, model.parameters(), create_graph=True,
    for grad in grad_params:
        grad_norm += grad.pow(2).sum()
    return grad_norm

def minimal_ratio(matrix):
    w, _ = np.linalg.eig(matrix)
    return (w > 0).mean()
```

conv2d是其中一個layer，所以要得到其中一個純量要從cnn.parameters()開始用5個for loop..

```
cnn = CNN()
para = cnn.parameters()
for p in para:
    print(p.shape)
```

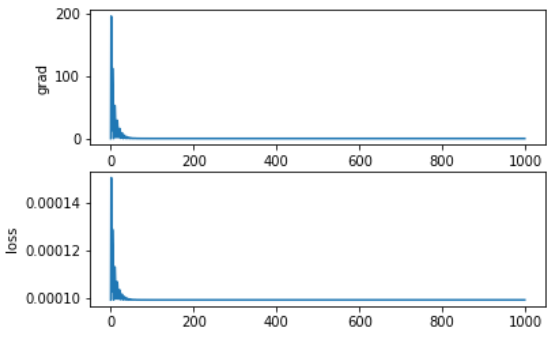
```
torch.Size([32, 1, 5, 5])
torch.Size([32])
torch.Size([32])
torch.Size([32])
torch.Size([10, 6272])
torch.Size([10])
```

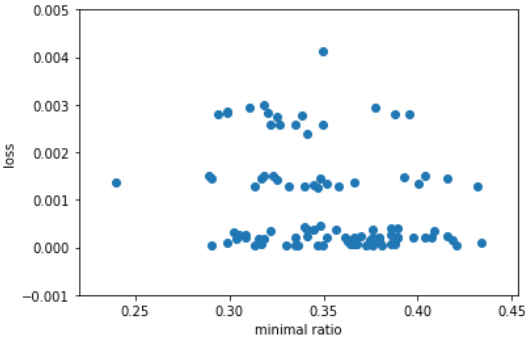
```
cnn = CNN()
para = cnn.parameters()
for pppp in para:
    print(pppp.shape)
    for ppp in pppp:
        print(ppp.shape)
        for pp in ppp:
            print(pp.shape)
            for p in pp:
                print(p.shape)
                for last in p:
                    print(last.shape)
                    assert False
```

```
torch.Size([32, 1, 5, 5])
torch.Size([1, 5, 5])
torch.Size([5, 5])
torch.Size([5])
torch.Size([1])
```

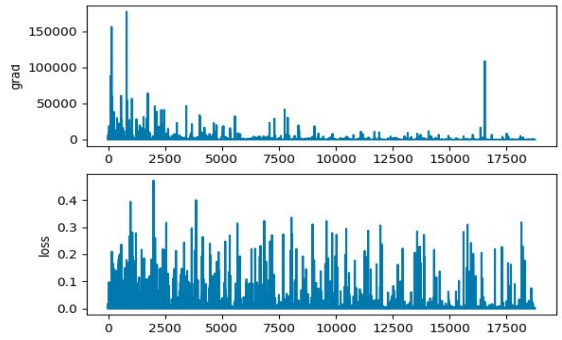
2. Train the model for 100 times. Plot the figure of minimal ratio to the loss. (2%)

 * Sinc function





 * MNIST

	因為mnist算hessian實在太耗時，所以就只有紀錄最小化gradient norm的過程

3. Comment your result. (1%)

   * Sinc function

   第一步先用loss function背向傳播之後，再換成用gradient norm背向傳播，訓練1000個epoch，learning rate設為1e-4。這一次因為是用梯度當作最小化的目標，經過數個epoch之後梯度就十分穩定的趨近於0，可以確保訓練完的權重們都是處在梯度為0的位置。重複訓練100次計算出的minimal ratio大致落在0.3~0.43這個區間，代表在高維的參數空間中，只有30%~40%的方向特徵值大於0，也就是說沿著這幾個方向移動會讓loss變大，因此該神經網路的參數只有在這30%~40%的方向上是local minimum。

   * 一開始覺得這個比例似乎也太小了，這不就表示其他方向上還有機會找到更低的loss嗎，不過觀察了一下發現特徵值為0的比例似乎也很多，算了一下幾乎要到50%，也就是總共加起來在80%~90%的方向上移動不會讓loss減少(如果不考慮更高階導數的話)，以這個task來說，應該算是可接受的local minimum。

```
w, _ = np.linalg.eig(hessian)
print("number of total eigenvalue: ", w.size)
print("number of zero eigenvalue:", (w == 0).sum())
print("zero ratio : %.2f%%" % ((w == 0).sum()/w.size * 100))
```

```
number of total eigenvalue:  609
number of zero eigenvalue: 289
zero ratio : 47.45%
```

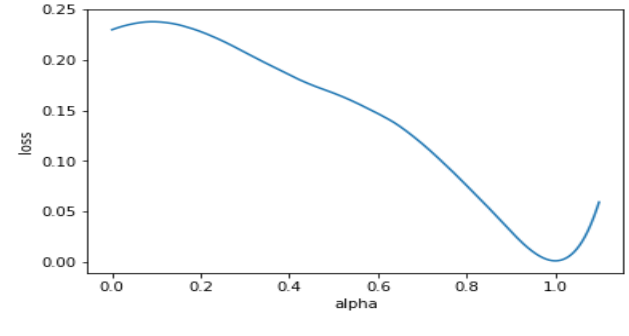
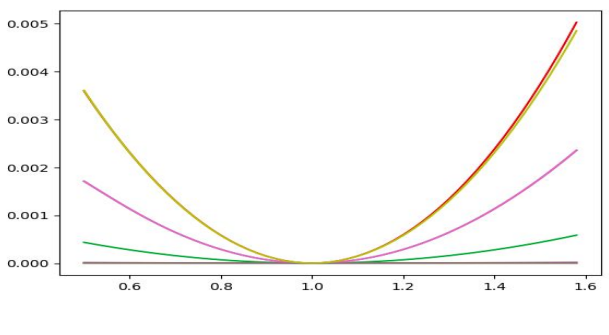
HW1-2(4)Bonus

1. Use any method to visualize the error surface/ Concretely describe your method and comment your result.

   * 觀察sinc function的error surface，α從0代表訓練前的神經網路，1代表訓練完的神經網路，而在其之間是兩者所有權重的內插，可以觀察到在訓練前後兩組參數的連線接近一開始的地方有一個稍微高起的障礙，所以以這個case來說，在訓練的一開始並不是筆直的朝α=1的參數前進，而是往其他方向繞開障礙。

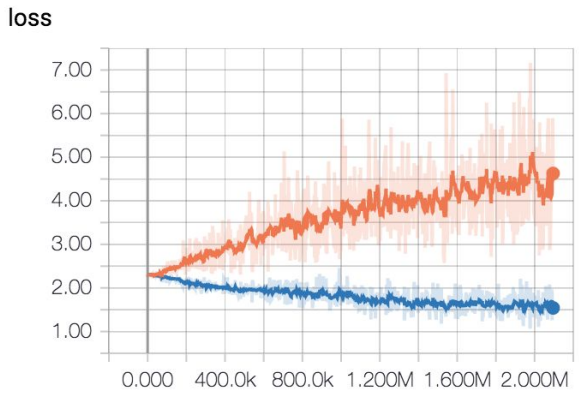
   接著是在訓練完的參數中隨機挑9個參數（訓練的神經網路共三層，一層挑三個），將參數個別乘上0.5~1.6，觀察單一參數改變時對loss的影響，結果可以看出訓練完的結果在每個被取樣的參數都是local minimum。

   * 結果：

HW1-3(1)Can network fit random variables?

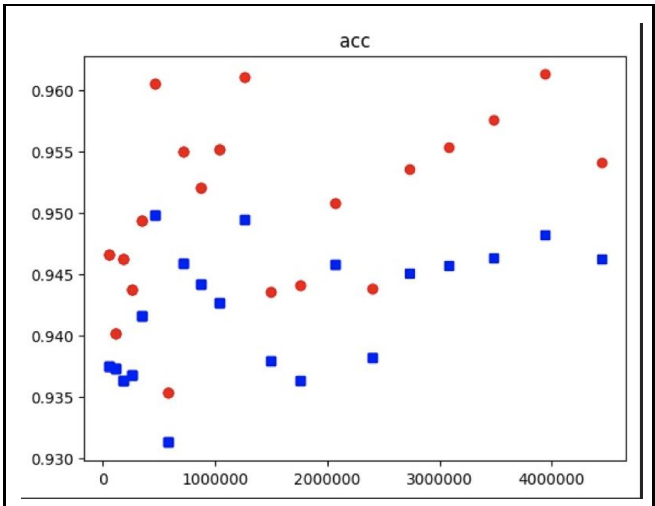
1. Describe your settings of the experiments. (e.g. which task, learning rate, optimizer) (1%)
   MNIST 上 ，用 3 hidden layer with 256 node，adam optimizer with 0.001 learning rate。
2. Plot the figure of the relationship between training and testing, loss and epochs. (1%)

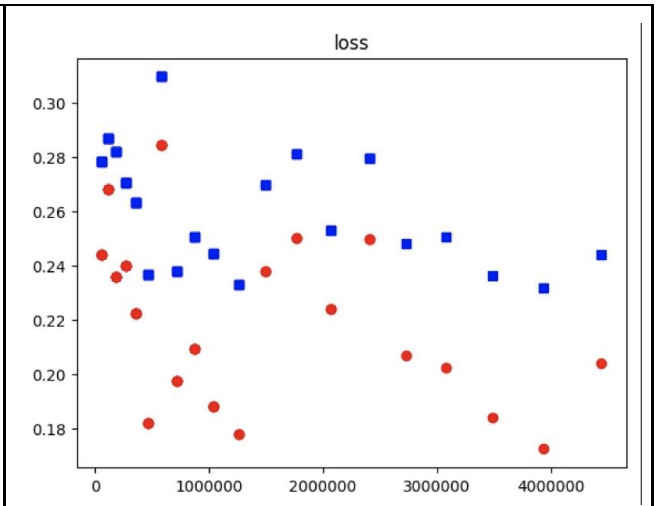


橘色是testing loss
藍色是training loss

HW1-3(2)Number of parameters v.s. Generalization

1. Describe your settings of the experiments. (e.g. which task, the 10 or more structures you choose) (1%)
   在MNIST上，我用兩層hidden layer 分別用 [64 ,128 ,256, 384, ,512] 個node，排列出所有可能，當loss在下個epoch之間，相差小於1e-3，視為收斂，停止training。
2. Plot the figures of both training and testing, loss and accuracy to the number of parameters. (1%)
   藍色代表 testing
   紅色代表 training



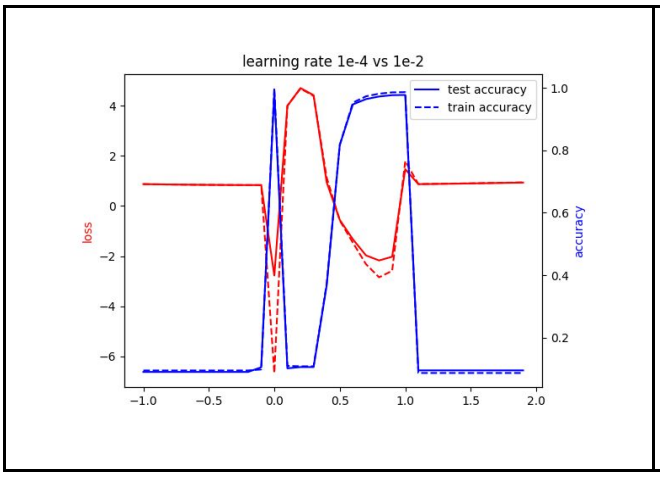


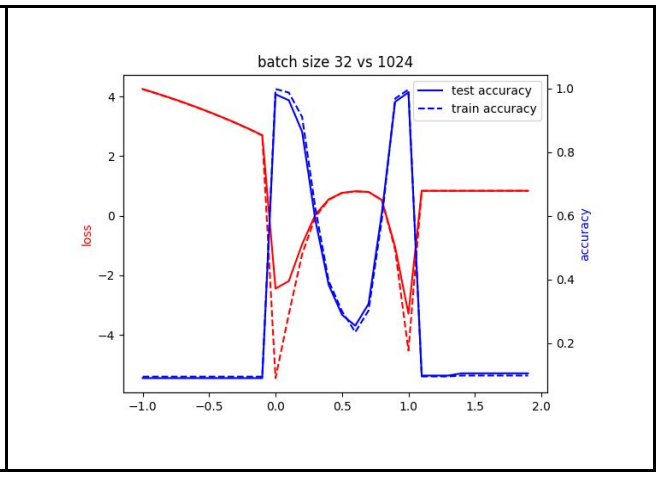
3. Comment your result. (1%)
   由點的分佈觀察，隨著numbers of parameters上升準確度有著越來越高的趨勢，但仔細觀察發現，在testing上，準確率最高的點在parameters數量接近10的6次方左右，代表該task在於這個parameters數量來說，訓練起來效率最好，parameters數量越大有漸漸發生overfitting的現象。從loss來看， 也是如此，數量越高loss越小，在於1e6 與 4e6有最低的loss。

HW1-3(3-1)Flatness v.s. Generalization Part I

1. Describe the settings of the experiments (e.g. which task, what training approaches) (0.5%)
   左圖是固定batch size為32，比較learning rate為1e-4（alpha=0）和1e-2（alpha=1），右圖是固定leaning rate為1e-4，比較batch size為32（alpha=0）和1024（alpha=1），loss為cross emtropy(log scale)。

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

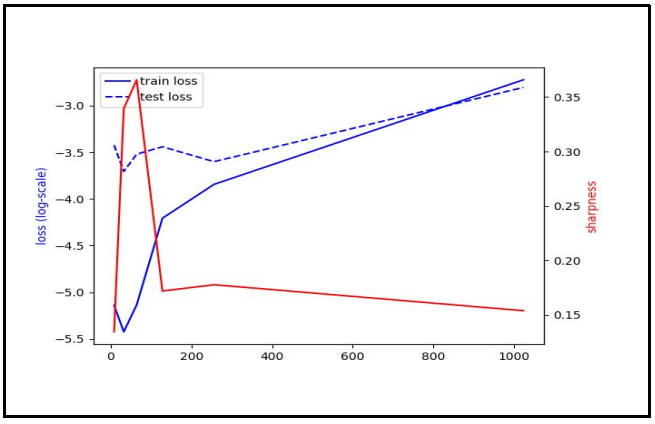




3. Comment your result. (1%)

HW1-3(3-2)Flatness v.s. Generalization Part II

1. Describe the settings of the experiments (e.g. which task, what training approaches) (0.5%)
   使用不同的batch size(8, 32, 64, 128, 256, 1024共六個)，去計算sharpness。這裡計算sharpness的方法為：先訓練好model，然後將所有的參數拉成一維，接著設定一個小範圍讓參數變動（例如每個值乘上1.05~0.95不等的隨機變數）然後sample 50個這樣的變動點，找出loss最大的，以 $\frac{\text{最大的}loss - \text{訓練的}loss}{\text{訓練的}loss}$ 當作sharpness。
2. Plot the figures of both training and testing, loss and accuracy, sensitivity to your chosen variable. (1%)



3. Comment your result. (1%)
   這個圖的大意是 batch size較小的情況（接近sgd）因為sharpness很高，所以model generalization的能力比較低，所以會看到test和train的差距很大。然後batchsize變大，雖然loss變多不過sharpness變小、test和train變接近了，驗證了sharpness和generalization的關係。

分工表：

葛竑志	**HW1-1 all**
塗是澈	**HW1-2 all, HW1-3-3**
孫盟強	**HW1-3-1, HW1-3-2**