Deep Learning (Homework 2)

Due date : 5/17/2019

1 Convolutional Neural Network for Image Recognition

In this exercise, you will construct a Convolutional Neural Network (CNN) for image recognition by using the Animals-10 dataset. The original dataset contains about 28K animal images with medium quality where there are 10 categories/animals. The 10 categories are *Dog*, *Horse*, *Elephant*, *Butterfly*, *Chicken*, *Cat*, *Cow*, *Sheep*, *Spider*, and *Squirrel*. Some example images of 10 categories are shown below.

i. Please describe in details how to preprocess the images in Animal-10 dataset with different resolutions and explain why. You have to submit your preprocessing code.

因為是第一次接觸Pytorch，在查資料的時候查到pytorch有一個能將多資料夾資料讀入的套件，叫做Imagefolder，本來希望將Train_x 、 Train_y、Test_x、Test_y存成處理好的變數，又去查了如何自定義Datasets的方法，在掙扎之後，發現**本次作業並不需要一定要將資料整理為Datasets**(因為不需要計算testing loss)，於是僅使用Imagefolder，將圖片讀入以做Training 跟Testing。

```
classes = ('Dog', 'Horse', 'Elephant', 'Butterfly', 'Chicken', 'Cat', 'Cow', 'Sheep', 'Spider', 'Squirrel')

transform = transforms.Compose(
        [transforms.Resize([128,128]),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train = torchvision.datasets.ImageFolder(root='./animal-10/train/', transform=transform)
train_loader = torch.utils.data.DataLoader(train, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)

val = torchvision.datasets.ImageFolder(root='./animal-10/val/', transform=transform)
val_loader = torch.utils.data.DataLoader(val, batch_size=len(val), shuffle=True, num_workers=2)
```

在處理的過程中理解到，使用這個Dataloader需要有一個大資料夾內含多個小資料夾，而小資料夾的名稱才可用來作為label，因此後續在做第一題的第三小題，各類別準確率的testing時，由於作業資料val資料夾中的各類別資料都為400筆，於是將batch_size設為400，且shuffle令為False，以解決沒有寫成datasets的小困擾。

另外學到的是，在pytorch中讀取資料有三步驟，第一個是定義了從何取資料、將資料整理成何種型態，第二是需要使用dataloader像是一個小手提包依照你設定的batch_size(手提包大小)來裝取圖片型態的資料，而最後讀取的時候需要以for迴圈讀取，**無法直接讀取dataloader這種承載資料的載體**。

ii. Please implement a CNN for image recognition. You need to design the network architecture , describe your network architecture and analyze the effect of different settings including stride size and filter/kernel size. Plot learning curve , classification accuracy of training and test sets as displayed in above figure.

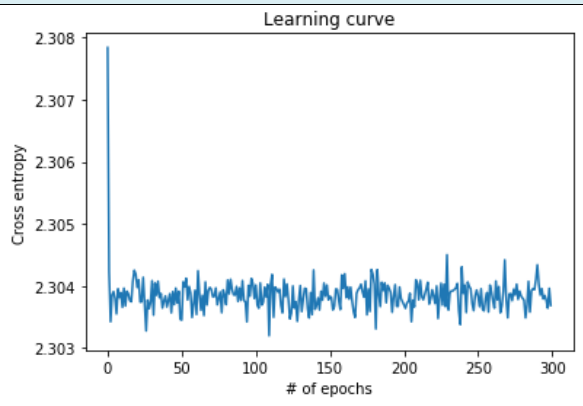
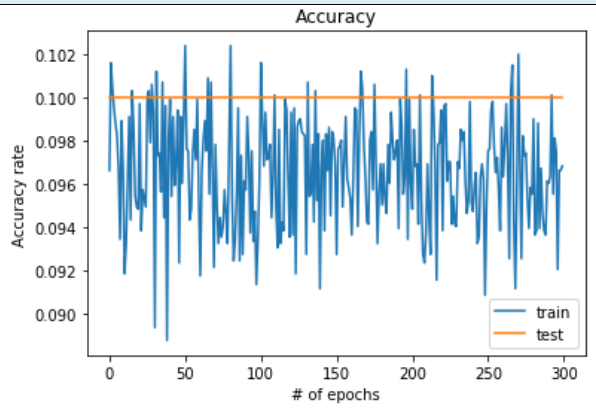
Q1-2 網路架構比較：

做了兩個不同的網路，第一個網路僅有兩層CNN，由於效果不佳於是做了第二個實驗，將網路加大。兩個實驗結果如下：

實驗一：兩層CNN

```
classifier_2(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=2048, out_features=10, bias=True)
)
```

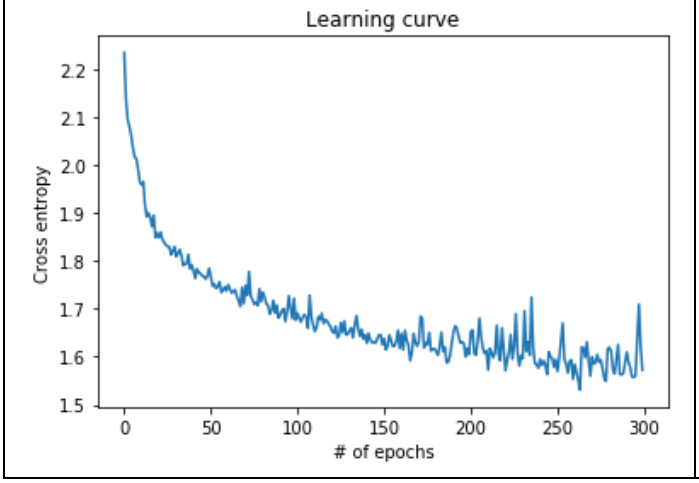
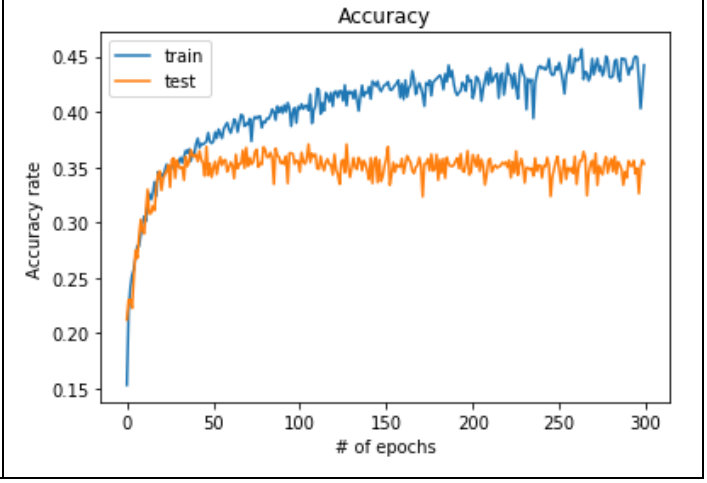
實驗一的Model架構只有兩層的CNN，因為想先接起一個簡單的Model拿來Training 看看，**效果不易外地很爛**。其他參數設定為： Resize 圖片的大小為32*32，用transforms 的 Normalize = ((0.5,0.5,0.5),(0.5,0.5,0.5)) 進行Normalize。Learning rate = 0.01, epoch = 300, batch_size = 64。

Learning Curve	Training & Testing Accuracy
	

實驗二：四層CNN

```
classifier(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=128, out_features=10, bias=True)
)
```

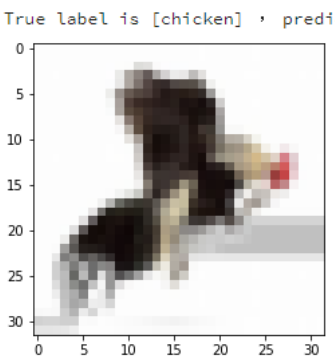
因為實驗一的效果實在太爛，連Training data 都 fit不起來，於是實驗二中加深網路，其餘實驗設定比照實驗一以做比較，效果比前次實驗好多了！跟其他同學討論以後，有同學說將Resize增大一些(介於32~256之間)會有更高的Accuracy。Resize 圖片的大小為32*32，用transforms 的 Normalize = ((0.5,0.5,0.5),(0.5,0.5,0.5))進行Normalize。Learning rate = 0.01, epoch = 300, batch_size = 64。（參數設定同實驗一）

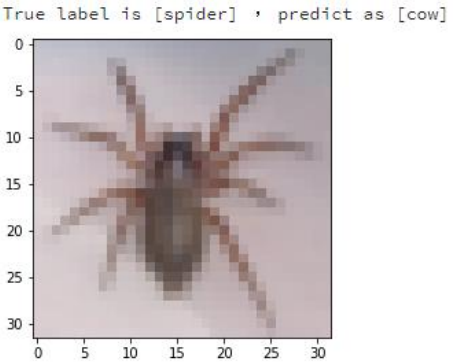
Learning Curve	Training & Testing Accuracy
	

iii. Show some examples of correctly and incorrectly classified images , list your accuracy for each test classes(similar to the following figure), and discuss about your results.

寫這題的時候就驗證第二小題的第二個實驗最後，根據同學的經驗圖片Resize大小大一些Training Accuracy能夠提升至六七成，原因是，像我resize成32*32，圖片看起來就太模糊了，不要說是機器判斷錯誤，連我自己都有可能判斷錯誤(看完機器判斷失誤的結果，感覺機器比我厲害多了哈哈哈)。

```
Accuracy of classes
butterfly : 51 %
cat : 45 %
chicken : 55 %
cow : 27 %
dog : 10 %
elephant : 30 %
horse : 36 %
sheep : 30 %
spider : 36 %
squirrel : 29 %
```





2 Recurrent Neural Network for Prediction of Paper Acceptance

In this exercise, you will implement a Recurrent Neural Network (RNN) model for Prediction of Paper Acceptance by using the machine learning conference papers from ICLR (International Conference on Learning Representations).

This dataset contains all the titles of the papers from ICLR 2017 and ICLR 2018. This dataset is separated into two files . ICLR_accepted.xlsx contains all the accepted papers, and ICLR_rejected.xlsx contains all the rejected papers. Please build the test data from these two files by using the first 50 papers from each of these two files.

Build your own dictionary and use the word embedding technique for data preprocessing. For example, given the sequence data "NCTU is good" and we can build a dictionary {0 : "NCTU", 1 : "is", 2 : "good"} . After the dictionary is built, we can convert the sequence to [0, 1, 2]. Then, we use the word embedding technique to deal with the sequence.

i. Please construct a standard RNN for acceptance prediction. For classification purpose, we aim to minimize the cross entropy error function, which is defined as

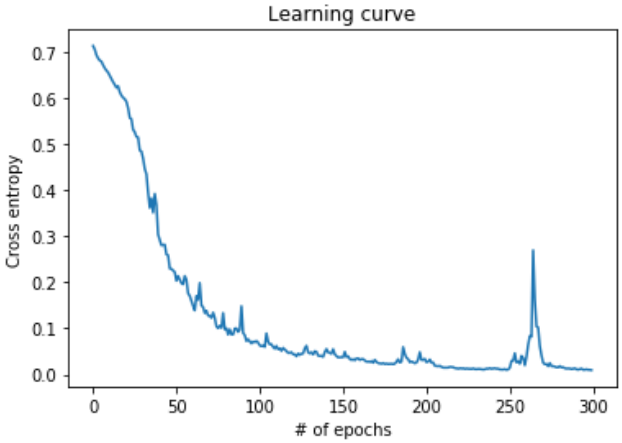
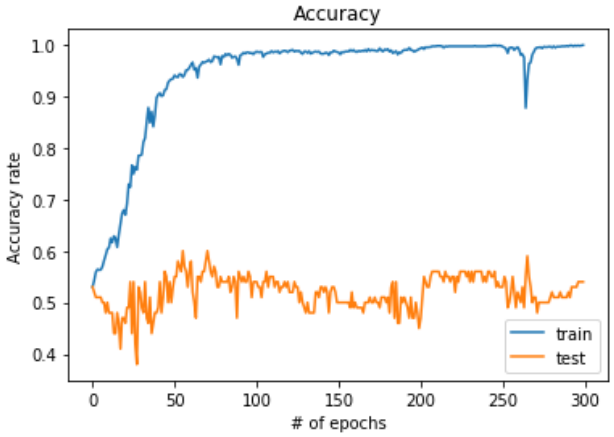
Design the network architecture by yourself, including number of hidden layers, number of hidden units, learning rate, number of iterations and mini-batch size. You have to show your (a)learning curve, (b)training error rate and (c)test error rate in the report.

```
RNN(
  (embed): Embedding(2421, 100)
  (rnn): RNN(100, 8)
  (out): Linear(in_features=8, out_features=2, bias=True)
)
```

使用RNN Model來做訓練，其中參數設定為EPOCH = 300，BATCH_SIZE = 64，TIME_STEP = 10， EMBED_SIZE = 100 ，LR = 0.001。LR是依照一般Adam的初始設定來設的。在做這個實驗的時候本來覺得nodes設多一點應該可以幫助訓練，於是一開始的時候hidden_size就直接設64，**出來的結果是training資料似乎已被model背起來**，導致testing accuracy一直沒有上升的趨勢。於是嘗試將網路縮小至hidden_size=8。

另外在做資料前處理的時候，將testing沒出現過的字和不足長度10的seq設成空白也就是0，做這部分的時候有兩大收穫，第一是當東西要塞入**embedding的時候shape需要經過transpose** 不然 train不起來，第二是，在**pytorch當中的label，會自動做成categorical_cross_entropy**，因此在最後Linear layer的output需要是兩個值，不能是一個。

```
Epoch:  280 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.48
Epoch:  281 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.48
Epoch:  282 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  283 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.48
Epoch:  284 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.48
Epoch:  285 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  286 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  287 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  288 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  289 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  290 | times: 0s | train loss: 0.0034 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  291 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  292 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  293 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  294 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  295 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  296 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  297 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  298 | times: 0s | train loss: 0.0032 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  299 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
Epoch:  300 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.47
```

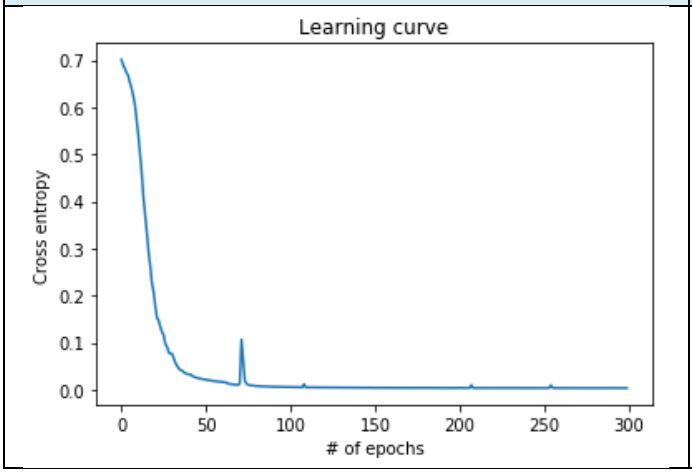
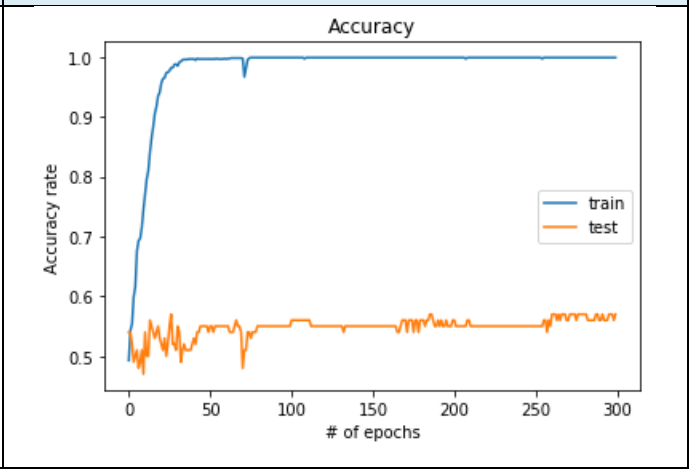
Learning Curve(RNN)	Training & Testing Accuracy(RNN)
	

ii. Redo I .by using the Long Short-Term Memory network (LSTM)

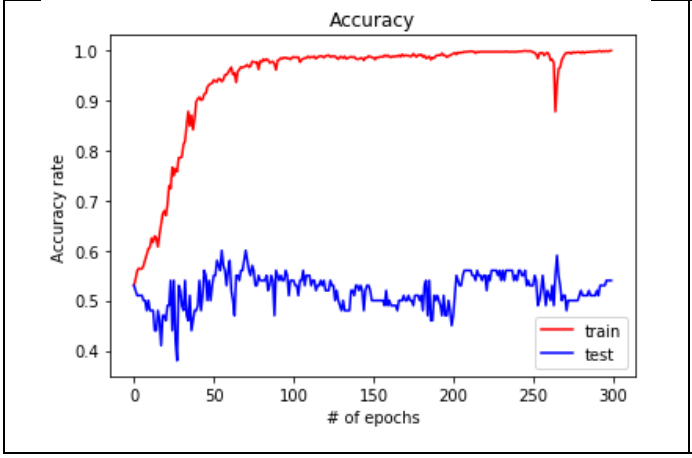
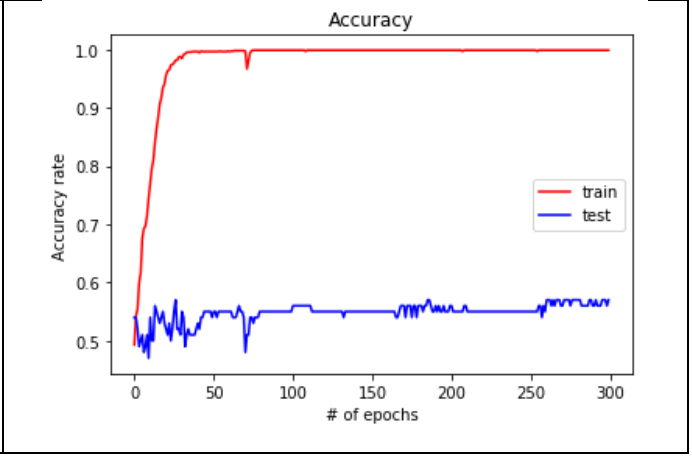
第二個實驗中，只是將原本的RNN模型改為LSTM，以方便做比較。

```
LSTM(
  (embed): Embedding(2421, 100)
  (lstm): LSTM(100, 8)
  (out): Linear(in_features=8, out_features=2, bias=True)
)
```

```
Epoch:  280 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  281 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  282 | times: 0s | train loss: 0.0034 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  283 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  284 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  285 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  286 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  287 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  288 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  289 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  290 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  291 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  292 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  293 | times: 0s | train loss: 0.0034 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  294 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  295 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  296 | times: 0s | train loss: 0.0034 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  297 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  298 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
Epoch:  299 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.56
Epoch:  300 | times: 0s | train loss: 0.0033 | train accuracy: 0.9992 | test accuracy: 0.57
```

Learning Curve(LSTM)	Training & Testing Accuracy(LSTM)
	

iii. Please discuss the difference between i. and ii .The following figure show an example of result.

RNN(Accuracy curve)	LSTM(Accuracy curve)
	

兩個實驗只是改了網路結構，從上圖中觀察可以看到**LSTM的成果比RNN的結果來得穩定**，不確定是不是因為LSTM的結構能夠讓模型保有短期記憶的效果，不像RNN的模型會將記憶區塊全部format的緣故。若從結果上來看的話，在上兩題黑色截圖中，可以看到RNN最後跑出來的結果約莫是在0.47,0.48 ，而LSTM則是在0.56,0.57，因此，**在本次實驗當中，LSTM較RNN有較高的準確度**。

另外的發現是，在嘗試實驗架構的時候，嘗試過網路架構64/32/16/8/4，其中網路並不是越大越好，而網路越小也不一定越好，也嘗試過把網路開的大一些，再加入dropout以避免model過度擬合 training data，而嘗試結果是，直接**把網路調小比加入dropout對於解決網路發生overfitting的狀況更有用**。