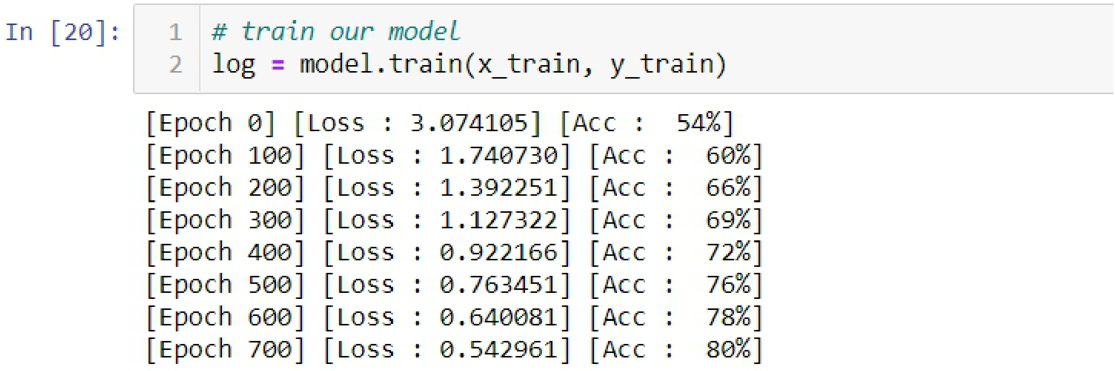
E14056114 徐顯舜 機械 109

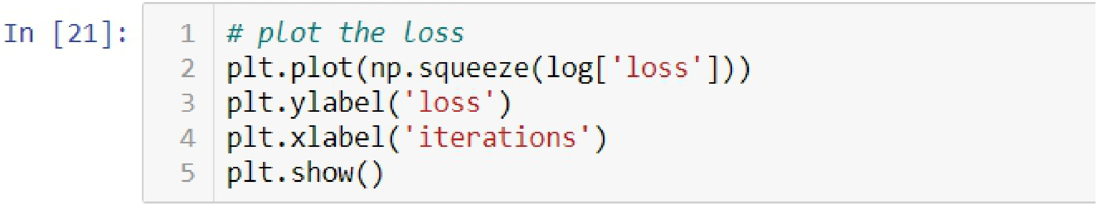
1.訓練過程截圖



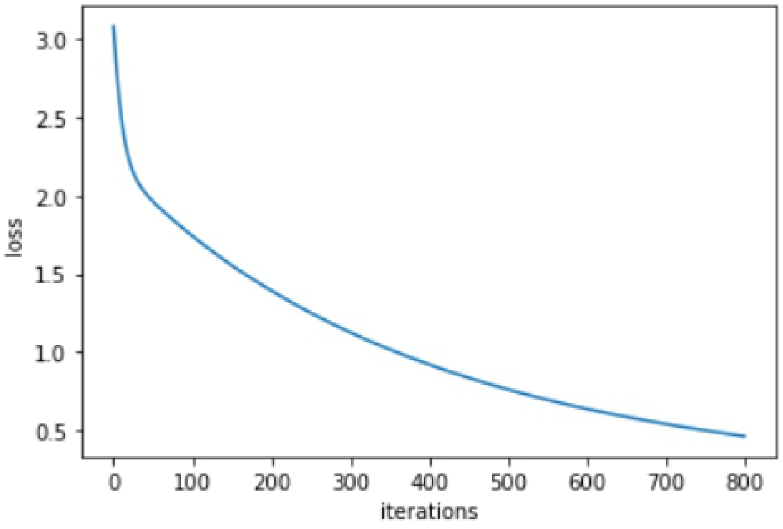
```
In [20]:
# train our model
log = model.train(x_train, y_train)
```

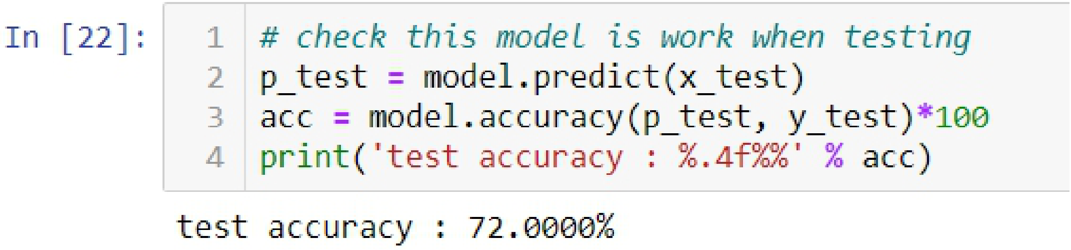
```
[Epoch 0] [Loss : 3.074105] [Acc :  54%]
[Epoch 100] [Loss : 1.740730] [Acc :  60%]
[Epoch 200] [Loss : 1.392251] [Acc :  66%]
[Epoch 300] [Loss : 1.127322] [Acc :  69%]
[Epoch 400] [Loss : 0.922166] [Acc :  72%]
[Epoch 500] [Loss : 0.763451] [Acc :  76%]
[Epoch 600] [Loss : 0.640081] [Acc :  78%]
[Epoch 700] [Loss : 0.542961] [Acc :  80%]
```

2.訓練結果



```
In [21]:
# plot the loss
plt.plot(np.squeeze(log['loss']))
plt.ylabel('loss')
plt.xlabel('iterations')
plt.show()
```





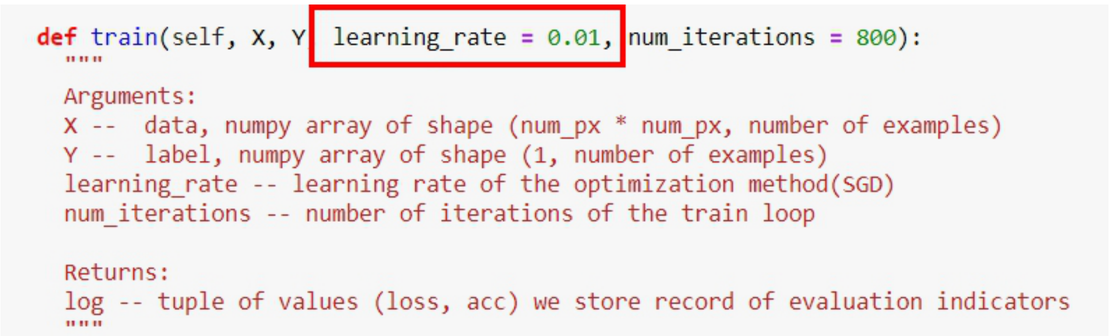
```
In [22]:
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)

test accuracy : 72.0000%
```

3.實驗及討論

因為這次的作業內容為判斷 mnist 手寫圖片數字為奇偶數(classification)，故使用 Logistic Regression 做二元分類，而其中適用的 activation function 為 sigmoid(輸出範圍[0,1])和 tanh(輸出範圍[-1,1])，但兩者的 Cross Entropy 不相同，要另外 define。但 tanh 不確定是否公式打錯因此無法 run，故這次程式碼修改主要在 learning rate、hidden layer 的 neurons 和 hidden layer 的層數。

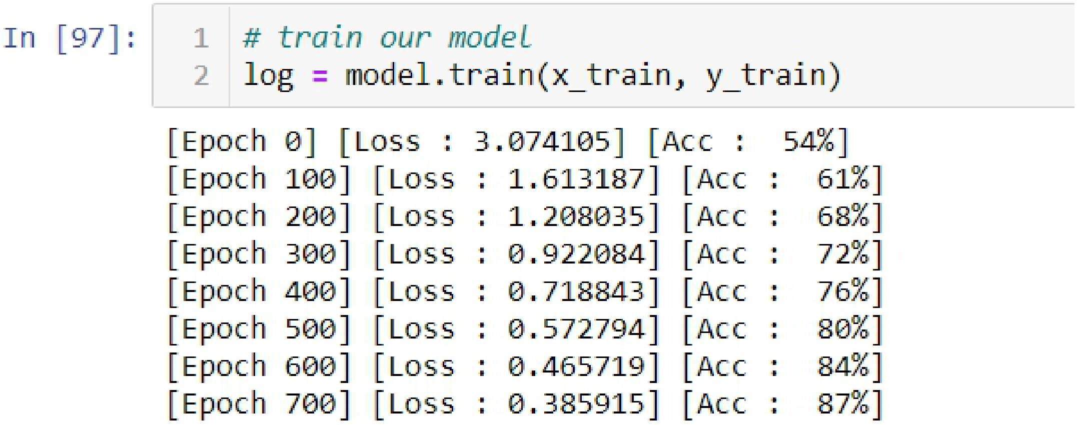
修改 Learning rate:

我們將 learning rate 提高至 0.01，實驗看看能不能提早收斂:



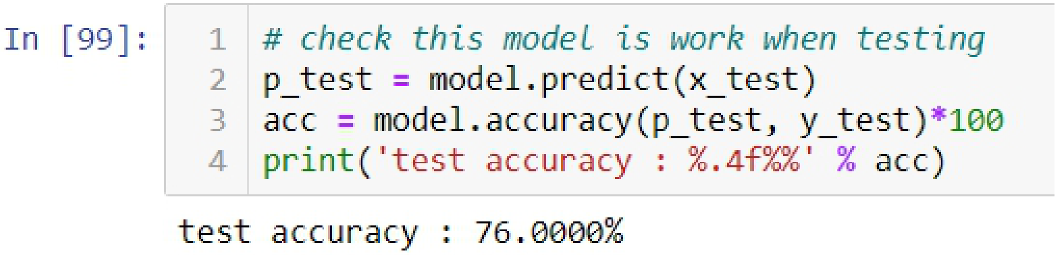
```
def train(self, X, Y  learning_rate = 0.01, num_iterations = 800):
    """
    Arguments:
    X --  data, numpy array of shape (num_px * num_px, number of examples)
    Y --  label, numpy array of shape (1, number of examples)
    learning_rate -- learning rate of the optimization method(SGD)
    num_iterations -- number of iterations of the train loop

    Returns:
    log -- tuple of values (loss, acc) we store record of evaluation indicators
    """
```

結果:



```
In [97]:
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 3.074105] [Acc :  54%]
[Epoch 100] [Loss : 1.613187] [Acc :  61%]
[Epoch 200] [Loss : 1.208035] [Acc :  68%]
[Epoch 300] [Loss : 0.922084] [Acc :  72%]
[Epoch 400] [Loss : 0.718843] [Acc :  76%]
[Epoch 500] [Loss : 0.572794] [Acc :  80%]
[Epoch 600] [Loss : 0.465719] [Acc :  84%]
[Epoch 700] [Loss : 0.385915] [Acc :  87%]
```

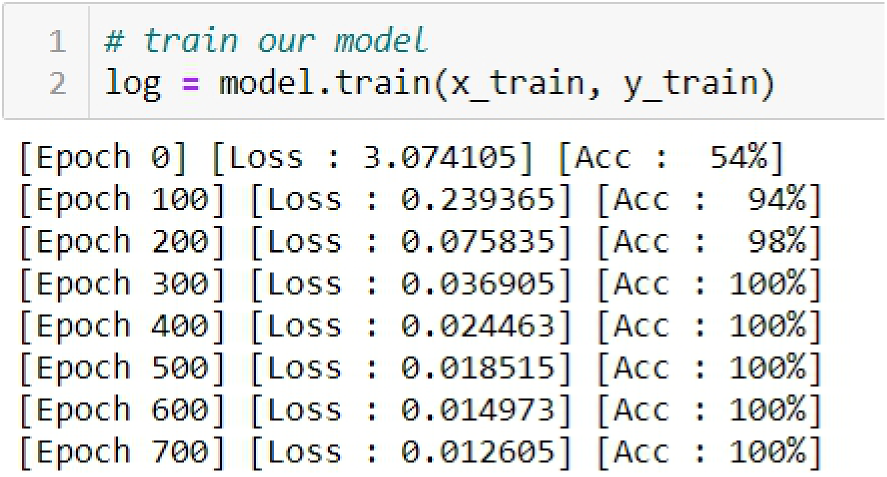


```
In [99]:
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)
```

```
test accuracy : 76.0000%
```

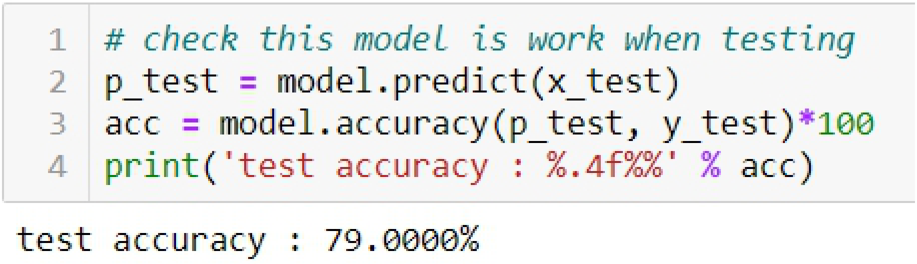
準確率上升至 76%，loss 收斂幅度稍微大一些。

將 learning rate 改成 0.1



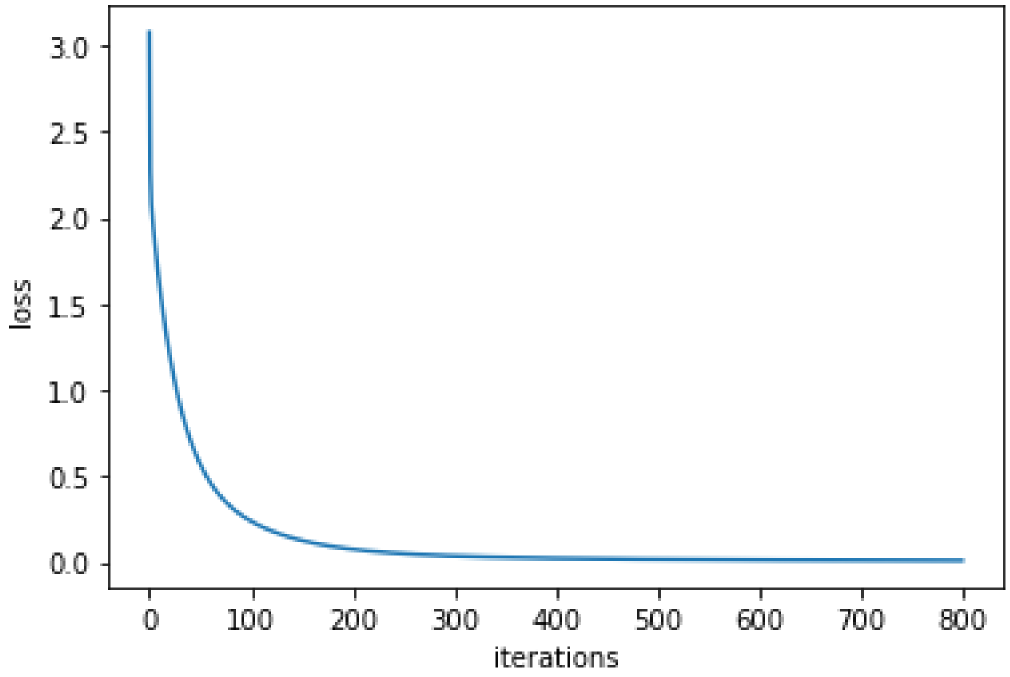
```
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 3.074105] [Acc :  54%]
[Epoch 100] [Loss : 0.239365] [Acc :  94%]
[Epoch 200] [Loss : 0.075835] [Acc :  98%]
[Epoch 300] [Loss : 0.036905] [Acc : 100%]
[Epoch 400] [Loss : 0.024463] [Acc : 100%]
[Epoch 500] [Loss : 0.018515] [Acc : 100%]
[Epoch 600] [Loss : 0.014973] [Acc : 100%]
[Epoch 700] [Loss : 0.012605] [Acc : 100%]
```



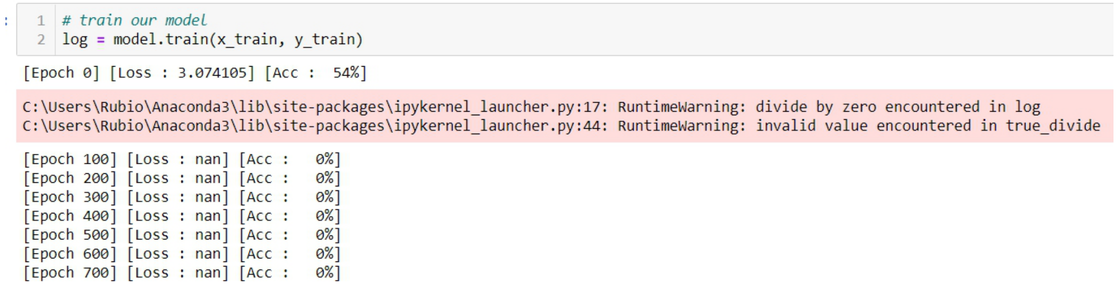
```
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)
```

```
test accuracy : 79.0000%
```



收斂的幅度更明顯，尤其從圖的曲線看，此時準確率達到 79%

繼續將 learning rate 加大至 0.53



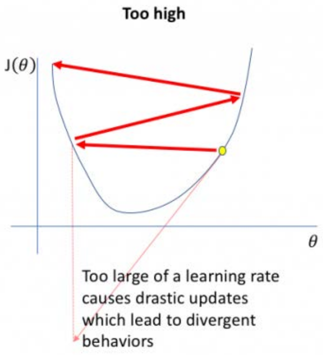
```
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 3.074105] [Acc :  54%]
```

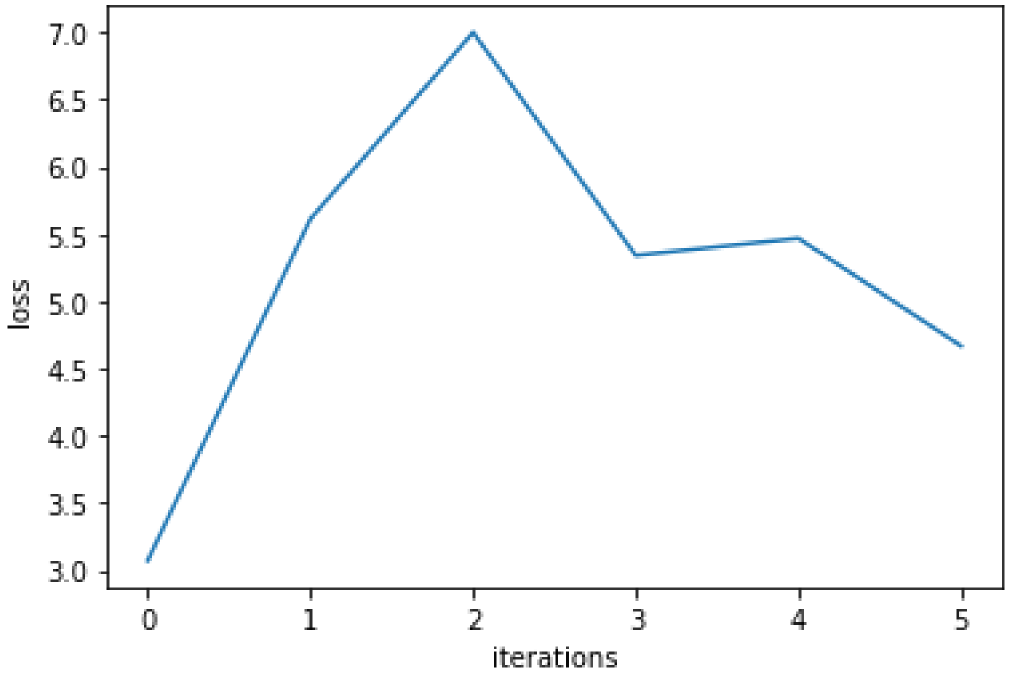
```
C:\Users\Rubio\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: RuntimeWarning: divide by zero encountered in log
C:\Users\Rubio\Anaconda3\lib\site-packages\ipykernel_launcher.py:44: RuntimeWarning: invalid value encountered in true_divide
```

```
[Epoch 100] [Loss : nan] [Acc :   0%]
[Epoch 200] [Loss : nan] [Acc :   0%]
[Epoch 300] [Loss : nan] [Acc :   0%]
[Epoch 400] [Loss : nan] [Acc :   0%]
[Epoch 500] [Loss : nan] [Acc :   0%]
[Epoch 600] [Loss : nan] [Acc :   0%]
[Epoch 700] [Loss : nan] [Acc :   0%]
```

此時發生與老師上課所講的，在學習過程中值修正的太多，造成無法收斂找到最小值，從 loss 與 iteration plot 更容易看出。

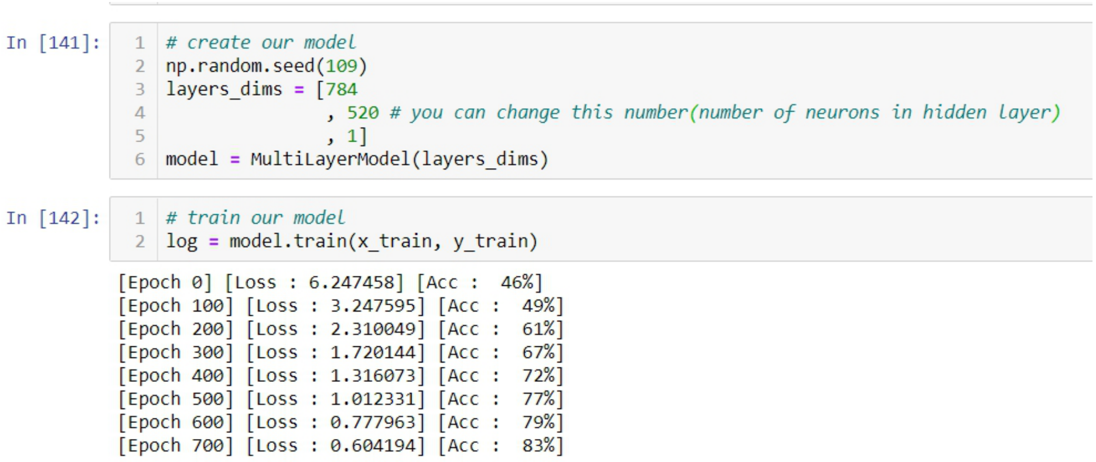


Too large of a learning rate causes drastic updates which lead to divergent behaviors



修改 hidden layer 的 neurons 數量

參考網路上的資訊，有人建議神經元的數量為小於 input 數量的 2/3，或者小於 1/2，因此我們先嘗試 2/3，將第二參數改為 520

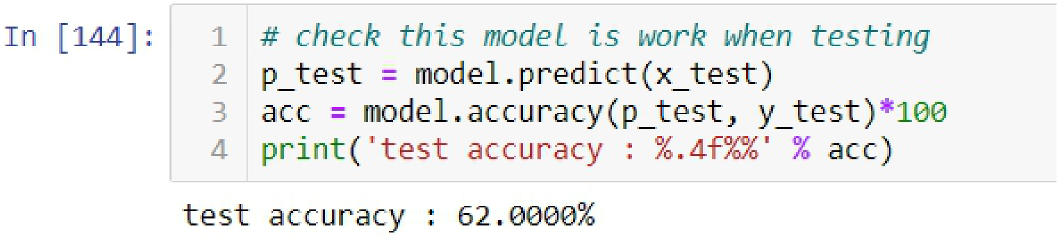

In [141]:

```
# create our model
np.random.seed(109)
layers_dims = [784
              , 520 # you can change this number(number of neurons in hidden layer)
              , 1]
model = MultiLayerModel(layers_dims)
```

In [142]:

```
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 6.247458] [Acc :  46%]
[Epoch 100] [Loss : 3.247595] [Acc :  49%]
[Epoch 200] [Loss : 2.310049] [Acc :  61%]
[Epoch 300] [Loss : 1.720144] [Acc :  67%]
[Epoch 400] [Loss : 1.316073] [Acc :  72%]
[Epoch 500] [Loss : 1.012331] [Acc :  77%]
[Epoch 600] [Loss : 0.777963] [Acc :  79%]
[Epoch 700] [Loss : 0.604194] [Acc :  83%]
```

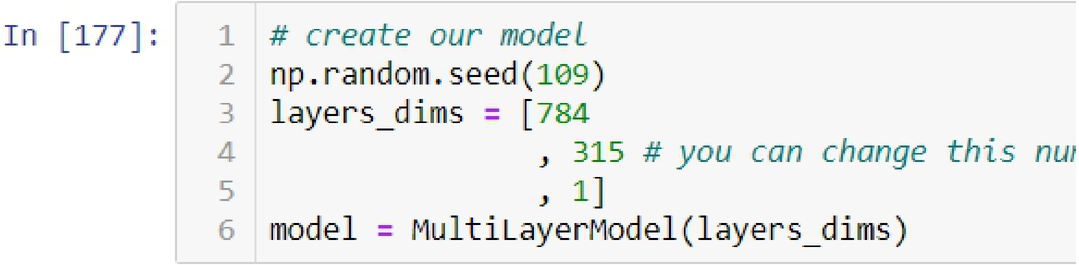

In [144]:

```
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)
```

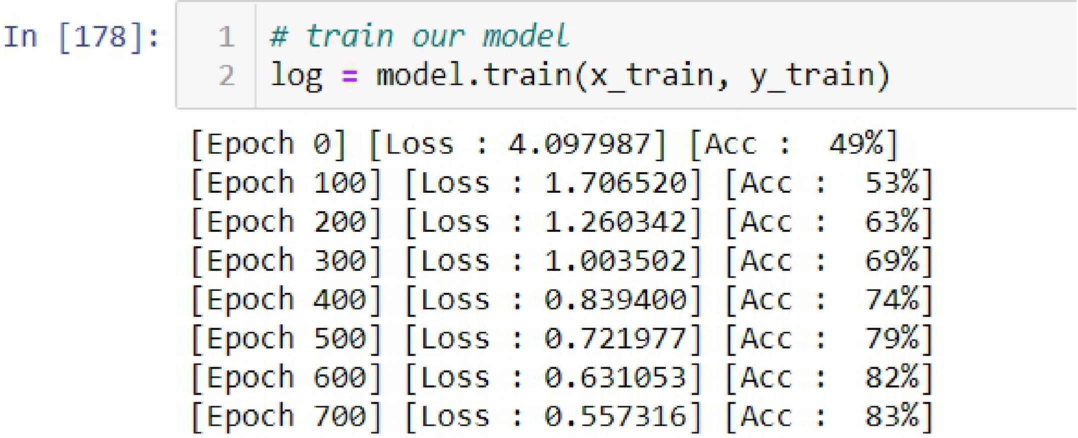
```
test accuracy : 62.0000%
```

發現 loss 的收斂數度更慢，造成整體的準確率下滑至 62%，執行過程中速度也明顯變慢。

我們嘗試 1/2，調整神經元的數量至 315

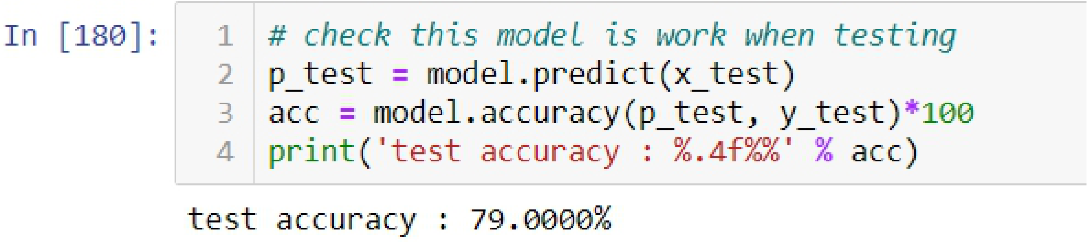


In [177]:
```
# create our model
np.random.seed(109)
layers_dims = [784
               , 315 # you can change this nu
               , 1]
model = MultiLayerModel(layers_dims)
```



In [178]:
```
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 4.097987] [Acc :  49%]
[Epoch 100] [Loss : 1.706520] [Acc :  53%]
[Epoch 200] [Loss : 1.260342] [Acc :  63%]
[Epoch 300] [Loss : 1.003502] [Acc :  69%]
[Epoch 400] [Loss : 0.839400] [Acc :  74%]
[Epoch 500] [Loss : 0.721977] [Acc :  79%]
[Epoch 600] [Loss : 0.631053] [Acc :  82%]
[Epoch 700] [Loss : 0.557316] [Acc :  83%]
```



In [180]:
```
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)
```

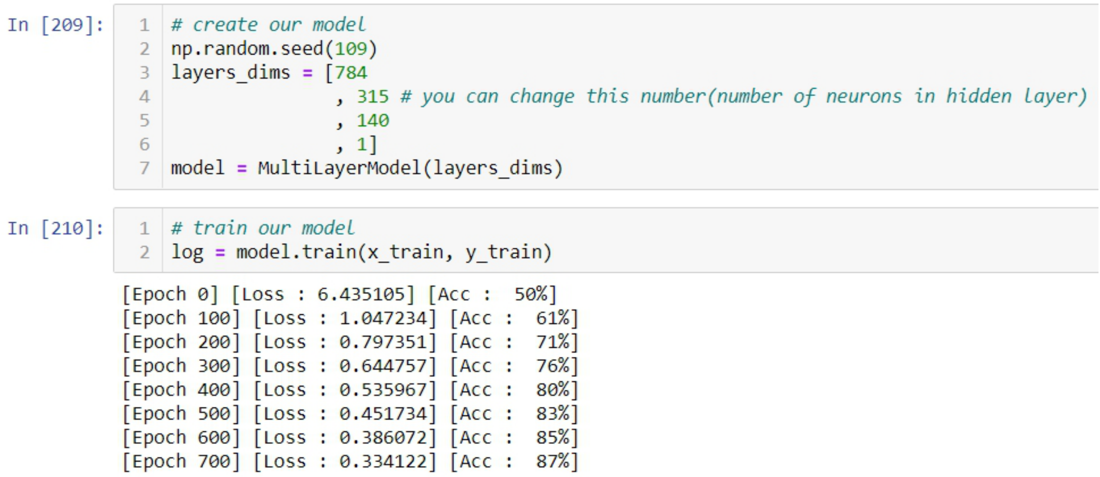
```
test accuracy : 79.0000%
```

發現雖然整體 loss 的大小比神經元 300 時的大，但反而在 test 的結果上是明顯優於後者，往後嘗試時也沒有發現有規律的法則去知道哪個神經元數量較好，也有可能是資料量不夠多訓練的緣故。

多增加幾層 hidden layer:

將 hidden layer 增加為兩層，而第二層的神經元數量設定小於第一層

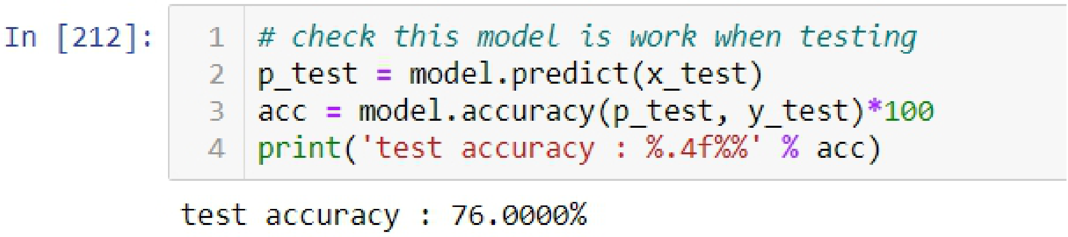
的 1/2



```
In [209]:
# create our model
np.random.seed(109)
layers_dims = [784
               , 315 # you can change this number(number of neurons in hidden layer)
               , 140
               , 1]
model = MultiLayerModel(layers_dims)
```

```
In [210]:
# train our model
log = model.train(x_train, y_train)
```

```
[Epoch 0] [Loss : 6.435105] [Acc :  50%]
[Epoch 100] [Loss : 1.047234] [Acc :  61%]
[Epoch 200] [Loss : 0.797351] [Acc :  71%]
[Epoch 300] [Loss : 0.644757] [Acc :  76%]
[Epoch 400] [Loss : 0.535967] [Acc :  80%]
[Epoch 500] [Loss : 0.451734] [Acc :  83%]
[Epoch 600] [Loss : 0.386072] [Acc :  85%]
[Epoch 700] [Loss : 0.334122] [Acc :  87%]
```



```
In [212]:
# check this model is work when testing
p_test = model.predict(x_test)
acc = model.accuracy(p_test, y_test)*100
print('test accuracy : %.4f%%' % acc)
```

```
test accuracy : 76.0000%
```

後續的 loss 雖然都比只有第一層的時候小，但得到的結果卻是準確率只有 76%；後續再多增加一層準確率只有 64%，因此並不是越多層越好，或者是神經元數量越多越好。