# 機器學習 HW7

姓名:林士恩

學號:B043011031

系級:108 電機甲

# 一、原始程式碼:

```python
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU, PReLU, ELU, Activation
from keras.utils import np_utils

batch_size = 100    #一次訓練的data個數
nb_classes = 10      #判斷的種類
nb_epoch = 20       #訓練週期(走完所有data算一個epoch)

#########################################data transformation#############################################
#print(mnist.load_data())
#print(type(mnist.load_data()))
#print(len(mnist.load_data()))          #data目前為len = 2的tuple data[0]為trainning sets, data[1]為testing sets
(X_trainning, Y_trainning), (X_testing, Y_testing) = mnist.load_data()

#print(X_trainning.shape)      #X_trainning 60000x28x28
#print(Y_trainning.shape)
#print(X_testing.shape)        #X_testing 10000x28x28

X_trainning = np.reshape(X_trainning, (60000, 28**2)).astype('float32') / 255      #換成60000x764 且為float的矩陣, 並轉換精確度
X_testing = np.reshape(X_testing, (10000, 28**2)).astype('float32') / 255          #換成10000x764 且為float的矩陣, 並轉換精確度

#keras.utils.to_categorical(y, num_classes=None) => Converts a class vector (integers) to binary class matrix.
Y_trainning = np_utils.to_categorical(Y_trainning, nb_classes)        #把Y換成60000x10x1的vectors
Y_testing = np_utils.to_categorical(Y_testing, nb_classes)
#print(Y_trainning.shape)

#########################################data transformation#############################################


model = Sequential()        #construct a NN(sequential object)
#print(type(model))
#keras.models.Dense(..) Just your regular densely-connected NN layer
model.add(Dense(input_dim = 28**2, units = 500))     #hidden units = 500
model.add(ELU(alpha=1.0))
model.add(Dense(units=500))                 #第2層hidden layer
model.add(ELU(alpha=1.0))
model.add(Dense(units=500))
model.add(ELU(alpha=1.0))
model.add(Dense(units=10))                  #output layer
model.add(ELU(alpha=1.0))

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

history = model.fit(X_trainning, Y_trainning,
                    batch_size = batch_size, epochs = nb_epoch,
                    verbose = 1, validation_data = (X_testing, Y_testing))    #verbose = Verbosity mode

score1 = model.evaluate(X_trainning, Y_trainning, verbose = 1)   #用此模型來測試理想準確度
score2 = model.evaluate(X_testing, Y_testing, verbose = 1)    #用此模型來測試理想準確度
print('epochs: %d, batch size: %d' %(nb_epoch, batch_size))
print('Test ideal cost:', score1[0])
print('Test ideal accuracy:%f ' %(score1[1] * 100))
print('Test actual cost:', score2[0])
print('Test actual accuracy:%f ' %(score2[1] * 100))
```

# Console:

```
epochs: 5, batch size: 100
Test ideal cost: 0.0382324556195
Test ideal accuracy:98.768333
Test actual cost: 0.0922312221728
Test actual accuracy:97.320000
```

# 二、問題討論:

## 1. 調整 activiation function 對於訓練模型的影響:

### 以下為所有測試結果:

### 1. 針對 ELU:

```
epochs: 5, batch size: 100
Test ideal cost: 0.0382324556195
Test ideal accuracy:98.768333
Test actual cost: 0.0922312221728
Test actual accuracy:97.320000
```

(hidden layer 為 ELU(alpha = 1.0),    output layer 為 softmax)

```
epochs: 5, batch size: 100
Test ideal cost: 0.0646679716475
Test ideal accuracy:98.018333
Test actual cost: 0.0963829775814
Test actual accuracy:97.140000
```

(hidden layer, output layer 皆為 sigmoid function)

```
epochs: 5, batch size: 100
Test ideal cost: 0.0519804593244
Test ideal accuracy:98.315000
Test actual cost: 0.0949512441518
Test actual accuracy:97.490000
```

(hidden layer 為 ELU(alpha = 1.0),    output layer 為 sigmoid)

```
epochs: 5, batch size: 100
Test ideal cost: 1.08637688041
Test ideal accuracy:56.680000
Test actual cost: 1.08327414465
Test actual accuracy:56.730000
```

(hidden layer, output layer 皆為 ELU(alpha = 1.0), output layer 為 softmax)

=>underfitting

```
epochs: 5, batch size: 100
Test ideal cost: 2.23095650915
Test ideal accuracy:80.070000
Test actual cost: 2.29520314112
Test actual accuracy:79.660000
```

(hidden layer, output layer 皆為 ELU(alpha = 1.0),)

```
epochs: 10, batch size: 100
Test ideal cost: 0.57274585921
Test ideal accuracy:91.163333
Test actual cost: 0.604255737233
Test actual accuracy:91.190000
```

(hidden layer, output layer 皆為 ELU(alpha = 1.0), epoch = 10)

```
epochs: 15, batch size: 100
Test ideal cost: 0.5002269319
Test ideal accuracy:90.045000
Test actual cost: 0.533115864742
Test actual accuracy:90.030000
```

(hidden layer, output layer 皆為 ELU(alpha = 1.0), epoch = 15)

```
epochs: 30, batch size: 100
Test ideal cost: 0.262173158556
Test ideal accuracy:95.213333
Test actual cost: 0.307285437382
Test actual accuracy:94.800000
```

(hidden layer, output layer 皆為 ELU(alpha = 1.0), epoch = 30)

從上列中的各種測試中，可以觀察到在 60000 筆手寫資料中，activiation function 使用 sigmoid function 與 ELU function 並沒有很大的差別，正確率都高達 97%，其中的原因在於這次的任務(辨識圖片)較簡單，沒辦法看出兩個激活函數的差別。其中也可以觀察到，若是把 ELU 中的 alpha 調大，會發現正確率大大地下降到 57%，從中可以發現 alpha 其實就像是以前做 gradient descent 中的 learning rate，若太大的話會導致 underfitting，造成結果不可靠。

並且，如果把 hidden layer, output layer 皆設為 ELU，會發現其實可以訓練得起來，正確率 80%，若再把 epoch 增大，讓他更多次數可以接近 optimal，可以發現正確率上升到了 94%，代表其實 ELU 可以當作 output layer 的激活函數，雖然效果沒有 softmax, sigmoid 來得好，因為他需要較多 epoch 來收斂，較不具效率。

## 2. 針對 ReLU:

```
epochs: 5, batch size: 100
Test ideal cost: 0.0319548323362
Test ideal accuracy:98.970000
Test actual cost: 0.0879355881586
Test actual accuracy:97.750000
```

(hidden layer 為 ReLU, output layer 為 sigmoid)

```
epochs: 5, batch size: 100
Test ideal cost: 9.70954070562
Test ideal accuracy:35.585000
Test actual cost: 9.60638490753
Test actual accuracy:35.870000
```

(hidden layer, output layer 皆為 ReLU) =>根本不準確

```
epochs: 5, batch size: 100
Test ideal cost: 0.0302785966264
Test ideal accuracy:99.068333
Test actual cost: 0.0829678395364
Test actual accuracy:97.730000
```

(hidden layer 為 ReLU, output layer 為 softmax)

從上列中的各種測試中，也可以觀察到在 60000 筆手寫資料中，activiation function 使用 sigmoid function 與 ReLU function 或 ELU 並沒有很大的差別，正確率都高達 97%，其中的原因在於這次的任務(辨識圖片)較簡單，沒辦法看出各種激活函數的差別。

如果把 hidden layer, output layer 皆設為 ReLU，會發現沒辦法訓練起來，正確率只有少少的 35%，需要搭配其他適合 output layer 的激活函數才可以使用。

## 3. 針對 Leaky ReLU:

```
epochs: 5, batch size: 100
Test ideal cost: 0.0572928405677
Test ideal accuracy:98.130000
Test actual cost: 0.112292794576
Test actual accuracy:96.940000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 sigmoid)

```
epochs: 5, batch size: 100
Test ideal cost: 0.0600052679236
Test ideal accuracy:98.118333
Test actual cost: 0.110049057042
Test actual accuracy:96.960000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 softmax)

```
epochs: 5, batch size: 100
Test ideal cost: 14.526970549
Test ideal accuracy:9.871667
Test actual cost: 14.5385218414
Test actual accuracy:9.800000
```

(hidden layer , output layer 皆為 LeakyReLU(alpha = 0.3))

```
epochs: 5, batch size: 100
Test ideal cost: 1.93354423169
Test ideal accuracy:83.290000
Test actual cost: 1.96989550242
Test actual accuracy:83.050000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 ELU(1.0)) =>83%可能只是剛好而已

```
epochs: 5, batch size: 100
Test ideal cost: 3.81007996979
Test ideal accuracy:65.328333
Test actual cost: 3.77865561066
Test actual accuracy:65.470000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 ELU(1.0))

```
Epoch 6/10
60000/60000 [==============================] - 18s 295us/step - loss:
3.5890 - acc: 0.6623 - val_loss: 3.5669 - val_acc: 0.7086
Epoch 7/10
60000/60000 [==============================] - 18s 293us/step - loss:
3.6303 - acc: 0.6539 - val_loss: 3.6232 - val_acc: 0.7004
Epoch 8/10
60000/60000 [==============================] - 18s 292us/step - loss:
3.5079 - acc: 0.7145 - val_loss: 3.5628 - val_acc: 0.6857
Epoch 9/10
60000/60000 [==============================] - 18s 296us/step - loss:
3.5983 - acc: 0.6627 - val_loss: 3.5148 - val_acc: 0.7396
Epoch 10/10
60000/60000 [==============================] - 18s 299us/step - loss:
3.4729 - acc: 0.7089 - val_loss: 4.4243 - val_acc: 0.4105
60000/60000 [==============================] - 8s 130us/step
10000/10000 [==============================] - 1s 134us/step
epochs: 10, batch size: 100
Test ideal cost: 4.33313565578
Test ideal accuracy:42.080000
Test actual cost: 4.42432327347
Test actual accuracy:41.050000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 ELU(1.0), epoch = 10)

```
epochs: 20, batch size: 100
Test ideal cost: 4.06841502972
Test ideal accuracy:50.450000
Test actual cost: 4.10276313896
Test actual accuracy:49.560000
```

(hidden layer 為 LeakyReLU(alpha = 0.3) , output layer 為 ELU(1.0), epoch = 20)

```
epochs: 5, batch size: 100
Test ideal cost: 0.0258593136046
Test ideal accuracy:99.120000
Test actual cost: 0.0835505433345
Test actual accuracy:97.600000
```

(hidden layer 為 LeakyReLU(alpha = 0.01) , output layer 為 softmax)    => overfitting

```
epochs: 5, batch size: 100
Test ideal cost: 14.435097433
Test ideal accuracy:10.441667
Test actual cost: 14.4611550446
Test actual accuracy:10.280000
```

(hidden layer 為 LeakyReLU(alpha = 10) , output layer 為 softmax)    => underfitting

從上列中的各種測試中，也可以觀察到在 60000 筆手寫資料中，activiation function 使用 sigmoid function 與 ReLU function 或 ELU 或 LeakyReLU 並沒有很大的差別，正確率都高達 97%，其中的原因在於這次的任務(辨識圖片)較簡單，沒辦法看出各種激活函數的差別。

如果把 hidden layer, output layer 皆設為 ReLU，也會發現沒辦法訓練起來，正確率只有 10%，需要搭配其他適合 output layer 的激活函數才可以使用。如果 output layer 的激活函數使用 ELU 在 epoch = 5 之下有高達 83%的正確率，但是如果提高 epoch = 10 時，在最後第 9 次到第 10 次 epoch 過程中可以發現正確率從 74%掉到大約 42%，最後結果沒有收斂，沒辦法訓練起來，提高再多的 epoch 也沒有用。所以推斷 83%只是初始值取得剛好而已。

最後結論為，適合用來當作 output layer 的激發函數:sigmoid, softmax, (ELU 不確定)

適合用來當作 hidden layer 的激發函數:ELU, LeakyReLU, ReLU

## 2. 調整 layer 數量對於 sigmoid 的影響:

## 以下為所有測試結果:

```
epochs: 5, batch size: 100
Test ideal cost: 0.0689953602364
Test ideal accuracy:97.968333
Test actual cost: 0.105972294239
Test actual accuracy:96.730000
```

(5 層)

```
epochs: 5, batch size: 100
Test ideal cost: 0.0902901316316
Test ideal accuracy:97.355000
Test actual cost: 0.127813915311
Test actual accuracy:96.430000
```

(7 層)

```
epochs: 5, batch size: 100
Test ideal cost: 0.189879395061
Test ideal accuracy:94.751667
Test actual cost: 0.211574826166
Test actual accuracy:94.280000
```

(8 層)

```
epochs: 5, batch size: 100
Test ideal cost: 2.30188572159
Test ideal accuracy:11.236667
Test actual cost: 2.30175134811
Test actual accuracy:11.350000
```

(9 層)

```
epochs: 5, batch size: 100
Test ideal cost: 2.30219708417
Test ideal accuracy:11.236667
Test actual cost: 2.30215792313
Test actual accuracy:11.350000
```

(10 層)

```
epochs: 5, batch size: 100
Test ideal cost: 2.30205212835
Test ideal accuracy:11.236667
Test actual cost: 2.30198075905
Test actual accuracy:11.350000
```

(15 層)

由上面的結果可以觀察到，當 hidden layer, output layer 都為 sigmoid function 時，在低層數時不太影響結果，都有 95%以上，但到了一定的層數時，正確率會下降到只有 10%，代表 sigmoid 的 kill the gradient 缺點到了高層樹會變得越來越明顯，因為在做 backpropagation 時，越往前面一層，gradient 消失等於 0 的數量會越來越多，導致訓練結果不好。

## 3.(sigmoid)不做 feature scaling 的影響:

## 以下為所有測試結果(皆為 sigmoid 作為激發函數):

```
epochs: 5, batch size: 100
Test ideal cost: 0.205587913715
Test ideal accuracy:93.423333
Test actual cost: 0.212457791844
Test actual accuracy:93.270000
```

(5 層)

```
epochs: 5, batch size: 100
Test ideal cost: 0.256179929229
Test ideal accuracy:92.136667
Test actual cost: 0.261570061666
Test actual accuracy:91.900000
```

(7 層)

```
epochs: 5, batch size: 100
Test ideal cost: 0.249492276565
Test ideal accuracy:92.313333
Test actual cost: 0.259290927839
Test actual accuracy:92.060000
```

(8 層)

```
epochs: 5, batch size: 100
Test ideal cost: 1.15401074346
Test ideal accuracy:9.871667
Test actual cost: 1.16143093204
Test actual accuracy:9.800000
```

(9 層)

```
epochs: 5, batch size: 100
Test ideal cost: 2.30196705615
Test ideal accuracy:9.736667
Test actual cost: 2.30182490387
Test actual accuracy:9.820000
```

(10 層)

由上面的結果可以觀察到，資料若沒有經過 feature scaling 的話，正確率會較有做 feature scaling 的模型低，因為 feature scaling 不光是為了降低計算複雜度，當以 sigmoid 來當作 hidden layer 的激發函數時，feature scaling 能讓資料縮小，所以使得會造成 saturate 的狀況減少，使得正確率上升，由第 9 層也可以發現已經造成嚴重的 gradient 遺失，沒辦法訓練模型，代表教有做 feature scaling 還快產生嚴重的 gradient 的狀況，所以可以認定 feature scaling 會影響一個 NN 對於 gradient 遺失的對抗強度。