機器學習 HW8

姓名:林士恩

學號:B043011031

系級:108 電機甲

一、 原始程式碼:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Activation
from keras import backend as K
import numpy as np
np.random.seed(1337)  # for reproducibility

batch_size = 128
num_classes = 10
epochs = 5

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 input_shape=input_shape))
```

```
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3), kernel_initializer='he_normal'))       #second layer
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))  #壓縮data維度
model.add(Dropout(0.01))
model.add(Flatten())                        #以下之後都為DNN操作，所以才要攤平成vectors
model.add(Dense(128))                         #third layer
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.01))
model.add(Dense(num_classes, activation='softmax'))                #last layer

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score1 = model.evaluate(x_train, y_train, verbose=0)
score2 = model.evaluate(x_test, y_test, verbose=0)
print('epochs: %d, batch size: %d' %(epochs, batch_size))
print('Test ideal cost:', score1[0])
print('Test ideal accuracy:%f ' %(score1[1] * 100))
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

需注意的是 np.random.seed()影響 python 各個隨機數值，在這次作業需要把每一次的 theta initialization 固定才有比較意義

二、問題討論:

1.在 DNN、CNN 之下，有無 batch normalization 的影響:

本次訓練作業的weights initialization皆固定(每次訓練參數層數相同之下結果都會相同)

以下 model 全使用 DNN + sigmoid:

```
epochs: 30, batch size: 200
Test ideal cost: 0.00327805930349
Test ideal accuracy:99.881667
Test actual cost: 0.0993509534128
Test actual accuracy:97.840000
```

3 層 layers, 有使用 batch normaliazation

```
epochs: 30, batch size: 200
Test ideal cost: 0.0112949751909
Test ideal accuracy:99.598333
Test actual cost: 0.0957745201683
Test actual accuracy:97.890000
```

3 層 layers, 無使用 batch normaliazation

```
epochs: 5, batch size: 200
Test ideal cost: 0.0918399358049
Test ideal accuracy:97.286667
Test actual cost: 0.120084030162
Test actual accuracy:96.350000
```

3 層 layers, 無使用 batch normaliazation

```
epochs: 5, batch size: 200
Test ideal cost: 0.0346906626812
Test ideal accuracy:98.918333
Test actual cost: 0.0899943060997
Test actual accuracy:97.440000
```

3 層 layers, 有使用 batch normaliazation

```
epochs: 10, batch size: 200
Test ideal cost: 0.00873876756637
Test ideal accuracy:99.713333
Test actual cost: 0.072762573081
Test actual accuracy:98.120000
```

3 層 layers, 無使用 batch normaliazation

```
epochs: 10, batch size: 200
Test ideal cost: 0.0091680244562
Test ideal accuracy:99.733333
Test actual cost: 0.0641251096263
Test actual accuracy:98.310000
```

3 層 layers, 有使用 batch normaliazation

以下 model 全使用 DNN + ReLU + softmax:

```
epochs: 5, batch size: 200
Test ideal cost: 0.0440423485071
Test ideal accuracy:98.613333
Test actual cost: 0.114984703236
Test actual accuracy:97.160000
```

5 層 layers, 無使用 batch normaliazation

```
epochs: 5, batch size: 200
Test ideal cost: 0.0181823394042
Test ideal accuracy:99.390000
Test actual cost: 0.0872277845894
Test actual accuracy:97.600000
```

5 層 layers, 有使用 batch normaliazation

```
epochs: 50, batch size: 200
Test ideal cost: 0.000904318177826
Test ideal accuracy:99.973333
Test actual cost: 0.114827967556
Test actual accuracy:98.400000
```

5 層 layers, 無使用 batch normaliazation(無 overfitting)

```
epochs: 50, batch size: 200
Test ideal cost: 0.00259446147468
Test ideal accuracy:99.933333
Test actual cost: 0.0788282489732
Test actual accuracy:98.470000
```

5 層 layers, 有使用 batch normaliazation(無 overfitting)

以下 model 全使用 CNN + ReLU + softmax:

```
epochs: 5, batch size: 128
Test ideal cost: 0.00697076276885
Test ideal accuracy:99.830000
Test loss: 0.0311910139159
Test accuracy: 0.9894
```

4 層 layers, 無使用 batch normalization

```
epochs: 5, batch size: 128
Test ideal cost: 0.00239083929985
Test ideal accuracy:99.975000
Test loss: 0.0314459606407
Test accuracy: 0.9897
```

4 層 layers, 有使用 batch normalization

由上列結果可以觀察到在這次手寫辨識時，有沒有做 batch normalization 其實都差不多，有時候做了 batch normalization 的正確率較沒有做 batch normalization 差了一點(0.6 ~ 0.8%)，也可以發現，就算把層數加到 5 層，epoch 提高到 50 也沒辦法造成 overfitting，可想而知，表示這是手寫辨識資料分布很平均正常，正常之下不太可能 overfitting，所以 batch normalization 就派不上用場了。

2. 在 DNN、CNN 之下，有無 dropout 的影響:

以下 model 全使用 DNN + ReLU + softmax:

```
epochs: 2, batch size: 200
Test ideal cost: 0.06800541938
Test ideal accuracy:97.811667
Test actual cost: 0.0913738668481
Test actual accuracy:97.200000
```

3 層 layers, 無使用 dropout

```
epochs: 2, batch size: 200
Test ideal cost: 0.0866214535715
Test ideal accuracy:97.398333
Test actual cost: 0.097485835272
Test actual accuracy:96.930000
```

3 層 layers, 有使用 dropout

```
epochs: 5, batch size: 200
Test ideal cost: 0.0183662921797
Test ideal accuracy:99.400000
Test actual cost: 0.0647977052799
Test actual accuracy:98.110000
```

5 層 layers, 無使用 dropout

```
epochs: 5, batch size: 200
Test ideal cost: 0.0422393023815
Test ideal accuracy:98.686667
Test actual cost: 0.075012713272
Test actual accuracy:97.670000
```

5 層 layers, 有使用 dropout

以下 model 全使用 CNN + ReLU + softmax:

```
epochs: 5, batch size: 128
Test ideal cost: 0.00697076276885
Test ideal accuracy:99.830000
Test loss: 0.0311910139159
Test accuracy: 0.9894
```

4 層 layers, 無使用 dropout

```
epochs: 5, batch size: 128
Test ideal cost: 0.019896608381
Test ideal accuracy:99.375000
Test loss: 0.0305663011085
Test accuracy: 0.9902
```

4 層 layers, 有使用 dropout

由上列結果可以觀察到在這次手寫辨識時，相較於 batch normalization ，dropout 的影響比較明顯，也可以發現有使用 dropout 時，訓練後的正確率會較無 dropout 還低，但是與真正的測試正確率差不多。5 層之下且 DNN，有 dropout，訓練正確率與真實正確率差 1.189%；無 dropout 則相差 1.4329%。在 CNN 之下結果更明顯，4 層之下，有 dropout，訓練正確率與真實正確率差 0.355%，無 dropout 則相差 0.89%。會有這種結果是因為 dropout 的功能在於強化各個 layer units，降低過度依賴目前訓練資料的

可能，也就是將低 overfitting 的可能，所以使得訓練結果可能不會很高，但是真正測試時又跟訓練結果相差很少。

3.CNN 之下，其他參數的影響:

以下 model 全使用 CNN + ReLU + softmax:

```
epochs: 5, batch size: 128
Test ideal cost: 0.00697076276885
Test ideal accuracy:99.830000
Test loss: 0.0311910139159
Test accuracy: 0.9894
```

4 層 layers, 無使用 dropout, 無使用 batch normalization

```
epochs: 5, batch size: 128
Test ideal cost: 0.0158893850743
Test ideal accuracy:99.523333
Test loss: 0.0317764058092
Test accuracy: 0.9896
```

4 層 layers, 有使用 dropout, 有使用 batch normalization

可以發現 batch normalization 加上 dropout 的結果沒有多大的變化，代表這次資料分布漂亮，不太可能有 overfitting 的可能。

以下 model 全使用 CNN + ReLU + softmax + dropout:

```
epochs: 5, batch size: 128
Test ideal cost: 0.00858387716041
Test ideal accuracy:99.763333
Test loss: 0.0299660300263
Test accuracy: 0.9904
```

4 層 layers, 有使用 dropout(2 層 dropout 機率皆為 10%)

```
epochs: 5, batch size: 128
Test ideal cost: 2.3011753198
Test ideal accuracy:11.236667
Test loss: 2.30104067078
Test accuracy: 0.1135
```

4 層 layers, 有使用 dropout(2 層 dropout 機率皆為 99%)

```
epochs: 5, batch size: 128
Test ideal cost: 0.0562039004848
Test ideal accuracy:98.331667
Test loss: 0.0543152131581
Test accuracy: 0.9834
```

4 層 layers, 有使用 dropout(2 層 dropout 機率皆為 75%)

```
epochs: 5, batch size: 128
Test ideal cost: 0.00817128991488
Test ideal accuracy:99.760000
Test loss: 0.0326701359248
Test accuracy: 0.9892
```

4 層 layers, 有使用 dropout(2 層 dropout 機率皆為 1%)

可以發現在有使用 dropout 之下，如果 dropout rate 太大的話，造成只有一點點的 hidden units 有訓練到，所以有可能沒辦法訓練起來；若是 dropout rate 太小的話，反而對於 overfitting 沒有較好的抵抗力，施勇了 dropout 的意義，所以太大太小都不是很好的數值，挑一個中間值是一個非常值得探討的題目。