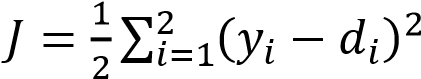
HW #7 Due: 6/7/2022

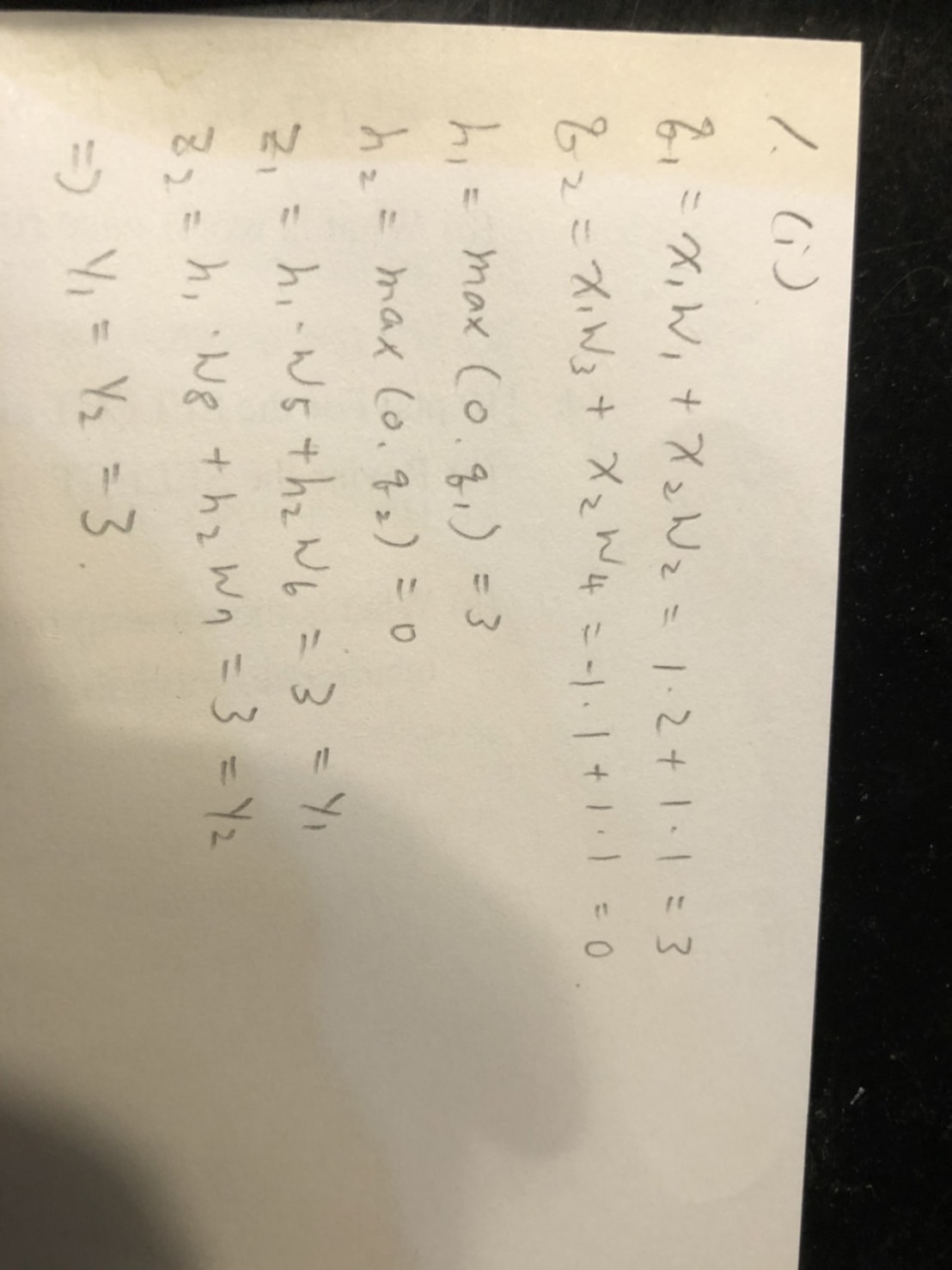
資工碩一 呂昀星 110598056

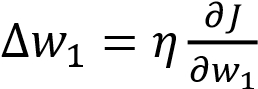
1. For the neural network given below, let 𝑤1 = 2.0, 𝑤2 to 𝑤8 be 1.0, 𝑑1 = 1.0,

𝑑2 = 0.0, 𝜂 = 0.1, 𝑥1 = 1.0, and 𝑥2 = −1.0. The activation function from 𝑞1 to ℎ1 and 𝑞2 to ℎ2 is ReLU, the activation function at the output nodes is linear (i.e., 𝑦 = 𝑧), and the cost function is

.

* 1. Find 𝑦1 and 𝑦2 (forward computation).



* 1. Find the value of  by using the BP algorithm.



*x*

2



*y*

1



*y*

2



*w*

3



*w*

2



*w*

4



*q*

1



*q*

2



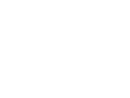
*w*

1



*x*

1



*w*

8



*w*

6



*w*

7



*z*

1



*z*

2



*w*

5



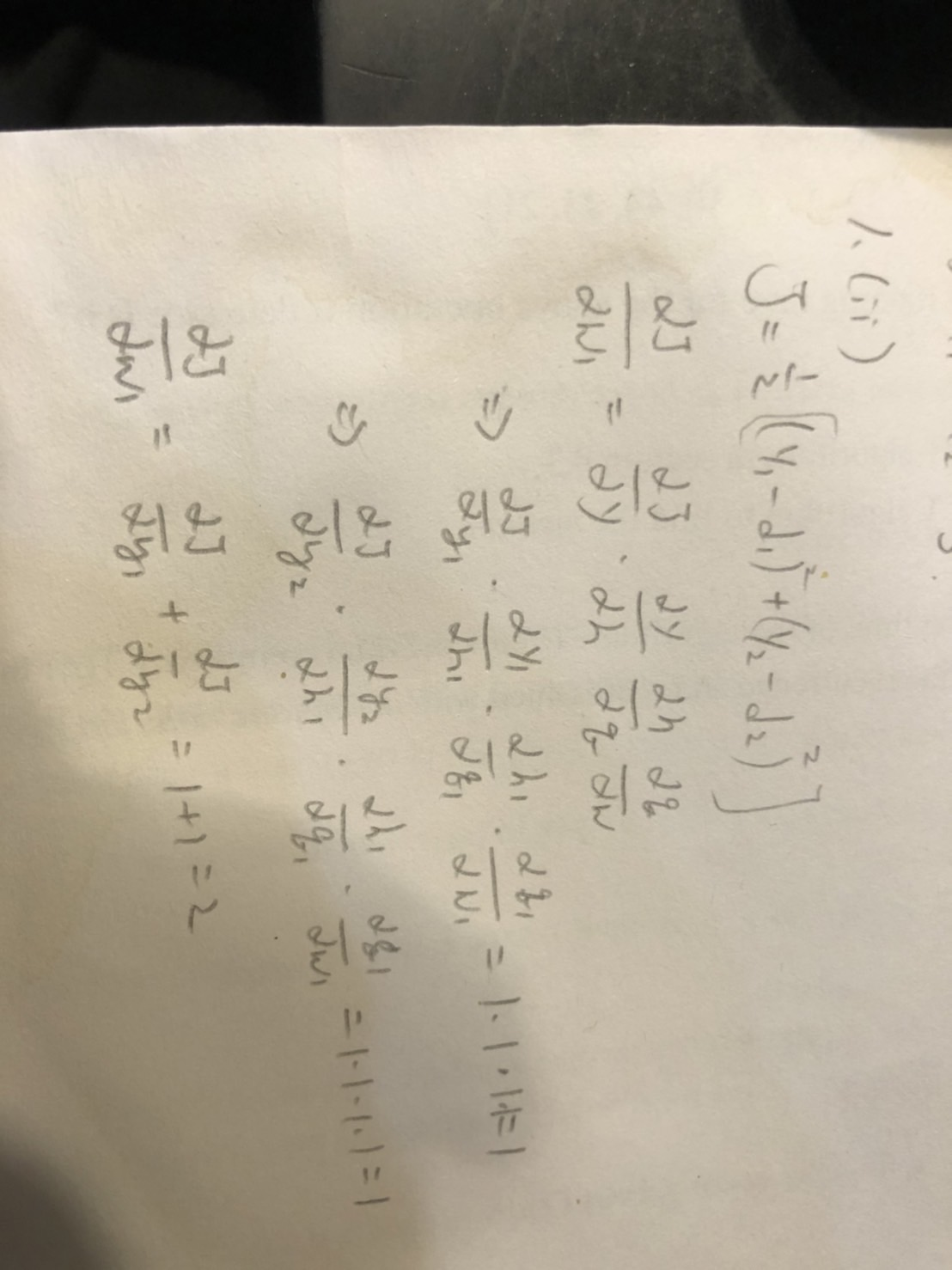
*h*

1

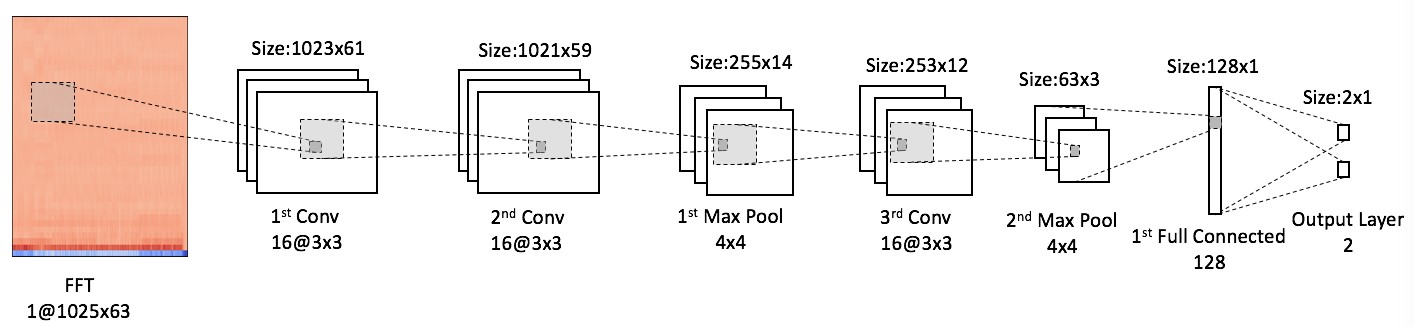


*h*

2



1. Compute the total number of *trainable* weights in the following CNN. To do so, you need to figure out the number of feature maps in each layer. This number is NOT equal to 3. To simplify the problem, ignore the bias terms. From the size of each feature map you should have no problem to figure out if zero-padding is used or not. Note: 16@3×3 means 16 kernels with size of 3×3.

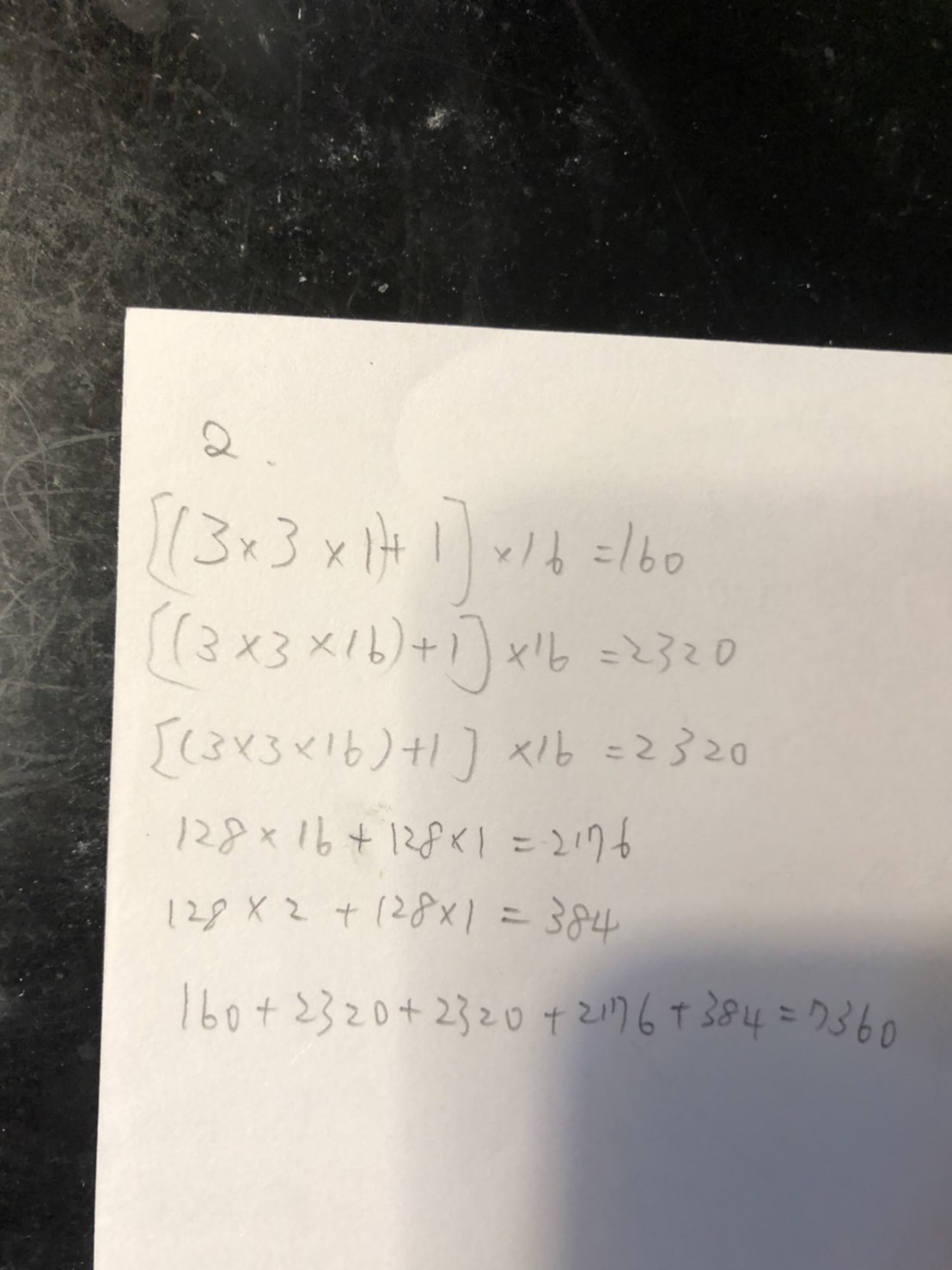


**x16**

**x16**

**Flatten layer inserted,**

**but not shown**



1. The CNN given in problem 2 is a binary classifier, and the activation function in the output layer is softmax. Because the dataset already contains all of the available labeled samples (several thousands), it is not possible to increase the

size of the dataset. Though the network has low training errors, unfortunately it has high validation errors. The followings are some possibilities considered to cope with this problem:

* 1. Change the network structure. If the hyper-parameters of only one layer are allowed to change, which layer should be considered first and how to change the hyper-parameters?
  2. Add/remove dropout layer. Will you add or remove dropout scheme during training? Why?

A:

Remove. Only use dropout layer in the testing.

* 1. Change the objective (loss) function. Suppose originally the objective function is cross entropy. Do you expect to obtain large accuracy improvement if the objective function is switched to MSE? Why?

A:

Yes, this is because it would be better to use cross entropy on binary classification problems and use MSE on CNN

1. Build a 3-layer neural network by busing Keras to classify the Iris dataset. Vary the hidden units from 10 to 100 in the 4 of 10 to observe the change of accuracy along with the number of hidden units. As usual, repeat the experiments 10 times to obtain the average accuracy. Use 10 epochs to train the network.

Accuracy : 0.9866666555404663

import tensorflow as tf

import pandas as pd

import numpy as np

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

iris = datasets.load\_iris()

data\_x=iris.data

data\_y=iris.target

data\_y= pd.get\_dummies(data\_y).values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(data\_x, data\_y, test\_size=0.2, random\_state=0)

model = tf.keras.Sequential([

tf.keras.layers.Dense(10, activation='relu'),

tf.keras.layers.Dense(100, activation='relu'),

tf.keras.layers.Dense(3, activation='softmax')

])

model.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

#avg accuracy for 10 times

loss\_sum=0

accuracy\_sum=0

for i in range(10):

model.fit(x\_train, y\_train, batch\_size=50, epochs=100)

loss, accuracy = model.evaluate(x\_test, y\_test, verbose=0)

loss\_sum+=loss

accuracy\_sum+=accuracy

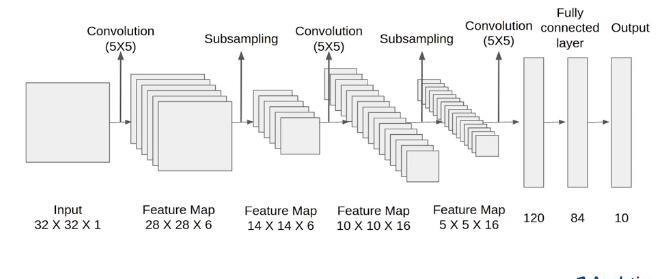
loss=loss\_sum/10.0

accuracy=accuracy\_sum/10.0

print ("Accuracy : ", accuracy)

print ("Loss : ", loss)

1. Build a modified LeNet-5, with original version shown below, by using Keras to classify the MNIST dataset. Note that the size of the images in the MNIST dataset is 28 x 28. The MNIST dataset is available in keras via tf.keras.datasets.mnist.load\_data().Use 20 epochs to train the CNN.



from \_\_future\_\_ import print\_function

import numpy as np

np.random.seed(1337)

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, LSTM,GRU

from keras.utils import np\_utils

from keras.callbacks import Callback

from keras.layers import Dense, Activation, Convolution2D, MaxPooling2D, Flatten

from keras.optimizers import Adam

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, f1\_score, precision\_score, recall\_score,accuracy\_score

# 寫一個LossHistory類，保存loss和acc

class LossHistory(Callback):

def on\_train\_begin(self, logs={}):

self.losses = {'batch':[], 'epoch':[]}

self.accuracy = {'batch':[], 'epoch':[]}

self.val\_loss = {'batch':[], 'epoch':[]}

self.val\_acc = {'batch':[], 'epoch':[]}

def on\_batch\_end(self, batch, logs={}):

self.losses['batch'].append(logs.get('loss'))

self.accuracy['batch'].append(logs.get('acc'))

self.val\_loss['batch'].append(logs.get('val\_loss'))

self.val\_acc['batch'].append(logs.get('val\_acc'))

def on\_epoch\_end(self, batch, logs={}):

self.losses['epoch'].append(logs.get('loss'))

self.accuracy['epoch'].append(logs.get('acc'))

self.val\_loss['epoch'].append(logs.get('val\_loss'))

self.val\_acc['epoch'].append(logs.get('val\_acc'))

def loss\_plot(self, loss\_type):

iters = range(len(self.losses[loss\_type]))

plt.figure()

# acc

plt.plot(iters, self.accuracy[loss\_type], 'r', label='train acc')

# loss

plt.plot(iters, self.losses[loss\_type], 'g', label='train loss')

if loss\_type == 'epoch':

# val\_acc

plt.plot(iters, self.val\_acc[loss\_type], 'b', label='val acc')

# val\_loss

plt.plot(iters, self.val\_loss[loss\_type], 'k', label='val loss')

plt.grid(True)

plt.xlabel(loss\_type)

plt.ylabel('acc-loss')

plt.legend(loc="upper right")

plt.savefig("mnist\_keras.png")

plt.show()

# 訓練參數

learning\_rate = 0.001

epochs = 20

batch\_size = 512

n\_classes = 10

# 定義圖像維度reshape

img\_rows, img\_cols = 28, 28

# 加載keras中的mnist數據集 分爲60,000個訓練集，10,000個測試集

# 將100張RGB，3通道的16\*32彩色圖表示爲(100,16,32,3)，（樣本數，高，寬，顏色通道數）

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# RNN shape

# x\_train = x\_train.reshape(-1, img\_rows, img\_cols)

# x\_test = x\_test.reshape(-1, img\_rows, img\_cols)

# CNN shape

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)

# 將X\_train, X\_test的數據格式轉爲float32

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

# 將X\_train, X\_test歸一化0-1

x\_train /= 255

x\_test /= 255

# 輸出0-9轉換爲ont-hot形式

y\_train = np\_utils.to\_categorical(y\_train, n\_classes)

y\_test = np\_utils.to\_categorical(y\_test, n\_classes)

# 建立模型

model = Sequential()

# lenet-5

model.add(Convolution2D(filters=6, kernel\_size=(5, 5), padding='valid', input\_shape=(img\_rows, img\_cols, 1), activation='tanh'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Convolution2D(filters=16, kernel\_size=(5, 5), padding='valid', activation='tanh'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(120, activation='tanh'))

model.add(Dense(84, activation='tanh'))

model.add(Dense(n\_classes, activation='softmax'))

#打印模型# verbose=1顯示進度條

model.summary()

model.compile(optimizer=Adam(lr=learning\_rate), loss='categorical\_crossentropy',metrics=['accuracy'])

history = LossHistory()

model.fit(x\_train, y\_train,batch\_size=batch\_size,epochs=epochs, verbose=1,validation\_data=(x\_test, y\_test),callbacks=[history])

model.save('rnn\_weight.h5')

# 測試

# model.load\_weights('rnn\_weight.h5')

y\_predict = model.predict(x\_test, batch\_size=512, verbose=1)

# y\_predict = (y\_predict > 0.007).astype(int)

y\_predict = (y\_predict > 0.01).astype(int)

y\_true = np.reshape(y\_test, [-1])

y\_pred = np.reshape(y\_predict, [-1])

# 評價指標

accuracy = accuracy\_score(y\_true, y\_pred)

precision = precision\_score(y\_true, y\_pred)

recall = recall\_score(y\_true, y\_pred, average='binary')

f1score = f1\_score(y\_true, y\_pred, average='binary')

# Micro F1: 將n分類的評價拆成n個二分類的評價，將n個二分類評價的TP、FP、RN對應相加，計算評價準確率和召回率，由這2個準確率和召回率計算的F1 score即爲Micro F1。

# Macro F1: 將n分類的評價拆成n個二分類的評價，計算每個二分類的F1 score，n個F1 score的平均值即爲Macro F1。

# 一般來講，Macro F1、Micro F1高的分類效果好。Macro F1受樣本數量少的類別影響大。

micro\_f1 = f1\_score(y\_true, y\_pred,average='micro')

macro\_f1 = f1\_score(y\_true, y\_pred,average='macro')

print('accuracy:',accuracy)

print('precision:',precision)

print('recall:',recall)

print('f1score:',f1score)

print('Macro-F1: {}'.format(macro\_f1))

print('Micro-F1: {}'.format(micro\_f1))

#繪製訓練的acc-loss曲線

history.loss\_plot('epoch')

accuracy: 0.992

