類神經網路 作業三\_Hopfield network 實作 大氣4A 黃展皇 106601015

工作環境：x64 windows10，conda 4.8.3，python3.6.10

1. 程式簡介、須包含實作架構(Hopfield)

程式碼如下所示，都應有足夠辨別之註解。

主要\_\_name\_\_ == '\_\_main\_\_'執行GUI介面規劃並執行main()。

main()作基本的防呆除錯並呼叫run\_hopfield()。

run\_hopfield()會針對不同資料集作訓練以及測試，並將每個測試結果印出。

而Hopfield network則由class hopfield\_network實現，具有train(輸入訓練集資料訓練網路權重)、w\_correction(計算權重調整量)、run(輸入測試集資料並回想)方法。

另有print\_results、load\_data方法幫助資料擷取轉型以及印出可視化資料。

import numpy as np

import os

import tkinter as tk

class hopfield\_network(object):

    def \_\_init\_\_(self, input\_num, n):

        self.input\_num = input\_num

        self.n = n

        self.w = np.zeros((n, n), dtype=np.float32)

    def train(self, data\_array):

        print('in training, check data\_array shape:', data\_array.shape)

        # every input loop, train w

        for i in range(self.input\_num):

            single\_data\_array = data\_array[i]

            single\_data\_array\_mean = float(single\_data\_array.sum()) / single\_data\_array.shape[0]

            self.w = self.w + self.w\_correction(single\_data\_array, single\_data\_array\_mean)

            # w diagonal line = 0

            for diagonal in range(self.n):

                self.w[diagonal, diagonal] = 0.0

        print('train success :)))\n')

    # calculate train weight correction

    def w\_correction(self, single\_data\_array, single\_data\_array\_mean):

        correction = np.zeros((self.n, self.n), dtype=np.float32)

        for i in range(self.n):

            correction[i] = (single\_data\_array - single\_data\_array\_mean)[i] \* (single\_data\_array - single\_data\_array\_mean)

        return correction / (self.n \* self.n \* single\_data\_array\_mean \* (1-single\_data\_array\_mean))

    def run(self, single\_data\_array):

        print('in testing, check single\_data\_array shape:', single\_data\_array.shape)

        for i in range(self.input\_num):

            u = self.w \* np.tile(single\_data\_array, (self.n, 1))

            ouput = u.sum(axis=1)

            # normalize

            m = float(np.amin(ouput))

            M = float(np.amax(ouput))

            ouput = (ouput - m) / (M - m)

            # to 0 or 1

            ouput[ouput <= 0.5] = 0.0

            ouput[ouput > 0.5] = 1.0

            return ouput

# input filename and return raw\_num, column\_num and data\_array(input\_num, raw\_num\*column\_num)

def load\_data(filename):

    # read file

    try:f = open(os.path.join(os.getcwd(), filename), mode='r')

    except OSError:f = open(os.path.join(os.getcwd(), 'hw3', filename), mode='r')

    origin\_str = f.read()

    f.close()

    # calculate raw\_num, column\_num and input\_num

    column\_num = len(origin\_str.split('\n')[0])

    raw\_num = int((len(origin\_str.split('\n\n')[0])+1)/(column\_num+1))

    input\_num = len(origin\_str.split('\n\n'))

    # create null data\_array(input\_num, raw\_num\*column\_num)(np.int)

    data\_array = np.zeros((input\_num, raw\_num\*column\_num), dtype=np.int)

    # input data\_array

    input\_count, rc\_count = 0, 0

    for s in origin\_str:

        if s == '\n':continue

        elif s == '1':

            data\_array[input\_count, rc\_count] = 1

            rc\_count += 1

        elif s == ' ':

            data\_array[input\_count, rc\_count] = 0

            rc\_count += 1

        else:print('s error!!!')

        if rc\_count == column\_num\*raw\_num:

            rc\_count = 0

            input\_count += 1

    print('data loaded, data\_array shape(input\_num, raw\_num, column\_num):', '({}, {}\*{})'.format(input\_num, raw\_num, column\_num))

    return raw\_num, column\_num, data\_array

def print\_results(raw\_num, column\_num, array):

    for raw in range(raw\_num):

        line\_str = ''

        for column in range(column\_num):

            if array[raw\*column\_num+column] == 1.0:line\_str += '\*'

            else:line\_str += ' '

        print(line\_str)

def run\_hopfield(basic\_bonus\_noise='Basic'):

    print()

    print('run\_hopfield init')

    # train

    \_, \_, training\_array = load\_data('{}\_Training.txt'.format(basic\_bonus\_noise))

    hnn = hopfield\_network(training\_array.shape[0], training\_array.shape[1])

    hnn.train(training\_array)

    # test and show

    raw\_num, column\_num, testing\_array = load\_data('{}\_Testing.txt'.format(basic\_bonus\_noise))

    for symbol in range(testing\_array.shape[0]):

        # prediction = every symbol prediction

        prediction = hnn.run(testing\_array[symbol])

        print('testing data:\n')

        print\_results(raw\_num, column\_num, testing\_array[symbol])

        print('prediction:\n')

        print\_results(raw\_num, column\_num, prediction)

    print('{} data ok!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n'.format(basic\_bonus\_noise))

# run when button be pushed, default = 'Basic'

def main():

    legal\_dataset\_name = ['Basic', 'Bonus', 'Noise']

    try:dataset\_name = dataset\_entry.get()

    except ValueError:dataset\_name='Basic'

    if dataset\_name not in legal\_dataset\_name:dataset\_name='Basic'

    run\_hopfield(basic\_bonus\_noise=dataset\_name)

# GUI interface

if \_\_name\_\_ == '\_\_main\_\_':

    print('init')

    window = tk.Tk()

    window.title('show you how hopfield network')

    window.geometry('500x200')

    window.configure(background='white')

    header\_label = tk.Label(window, text='ok就按按鈕執行吧:')

    header\_label.pack()

    dataset\_frame = tk.Frame(window)

    dataset\_frame.pack(side=tk.TOP)

    dataset\_label = tk.Label(dataset\_frame, text='要執行哪個dataset(default=Basic)')

    dataset\_label.pack(side=tk.LEFT)

    dataset\_entry = tk.Entry(dataset\_frame)

    dataset\_entry.pack(side=tk.LEFT)

    calculate\_btn = tk.Button(window, text='按下去開始算', command=main)

    calculate\_btn.pack()

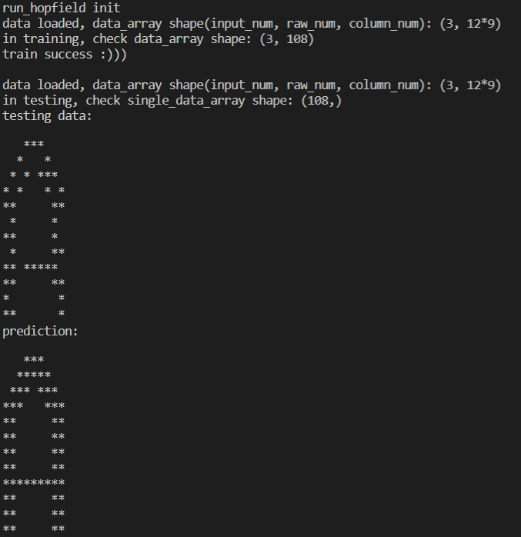
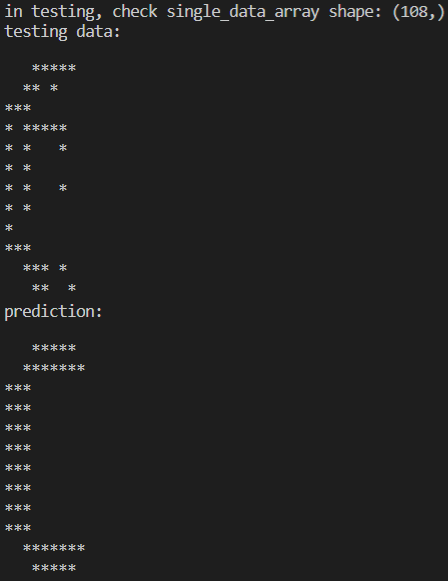
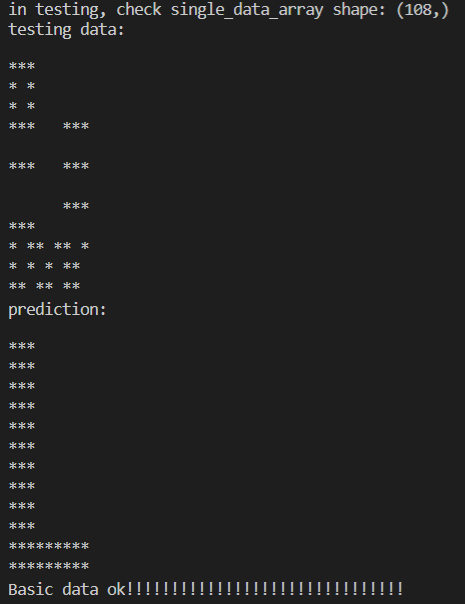
    window.mainloop()

1. 程式執行說明。(如何操作、使用)

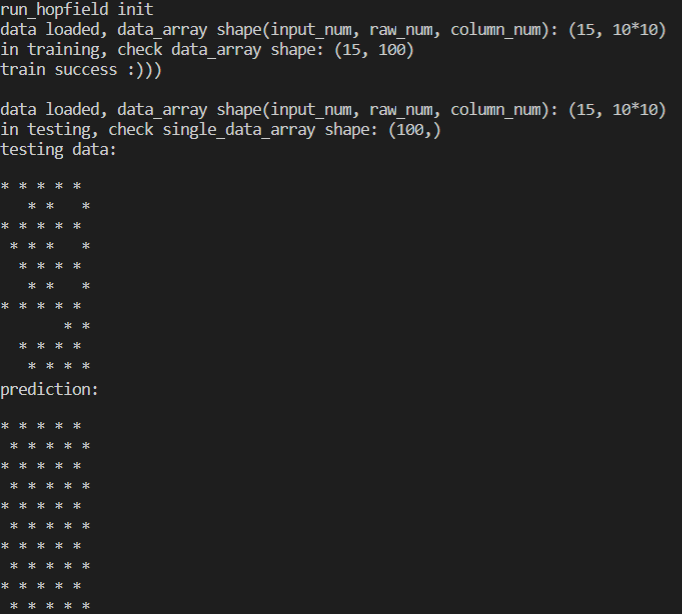
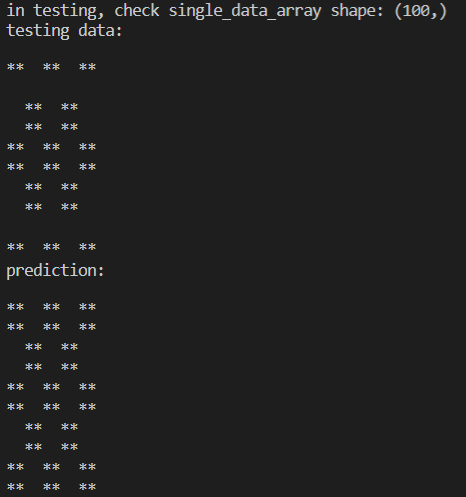
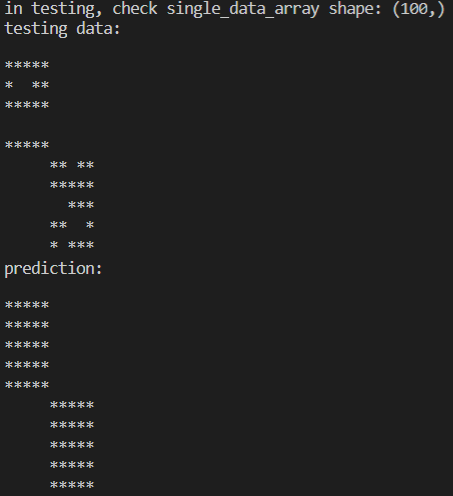
如影片所示，執行exe檔案會跑出一GUI介面，可選擇想要用以訓練及測試的資料集['Basic', 'Bonus', 'Noise']，預設及防呆都是'Basic'，即可在終端機看到精心設計的結果呈現；若直接執行.py檔案則同理，直接執行即可，一樣會有GUI介面可供應用。

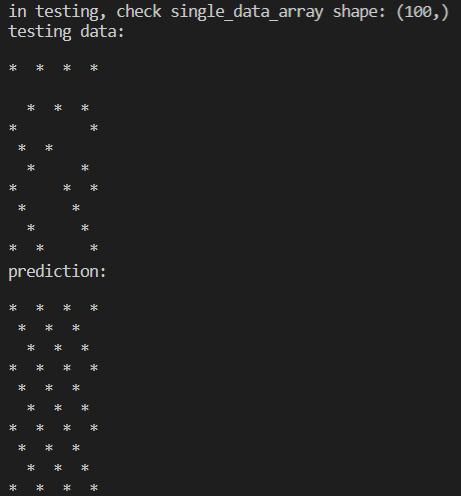
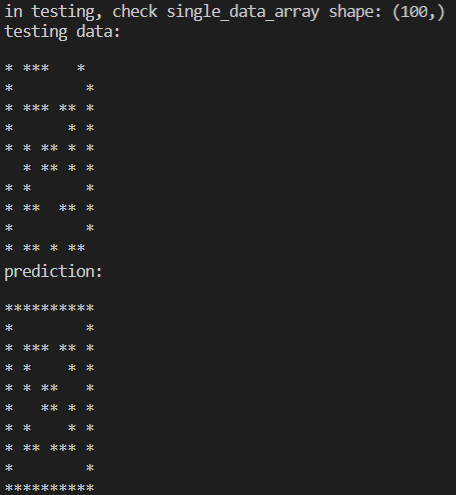
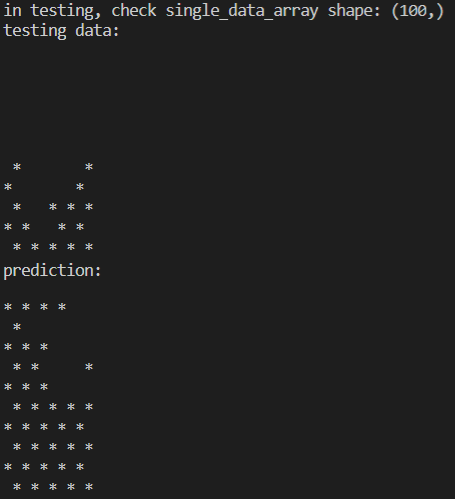
1. 實驗結果(所有資料集都須有實驗結果集說明)。

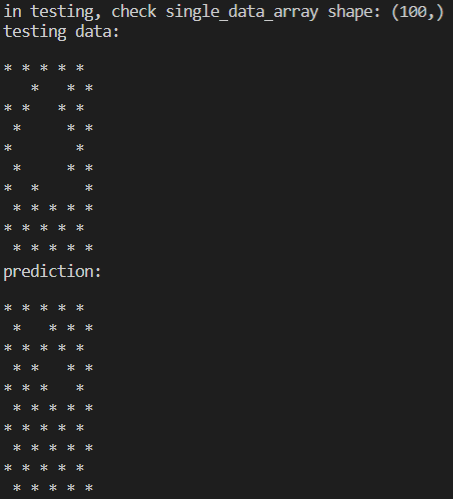
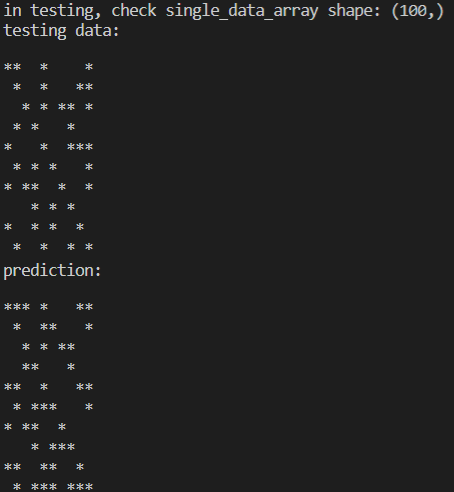
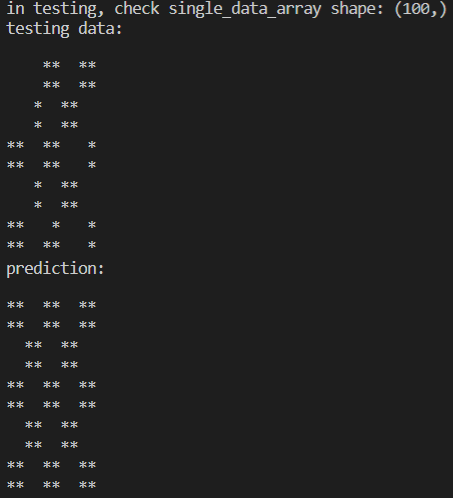
Basic資料集：

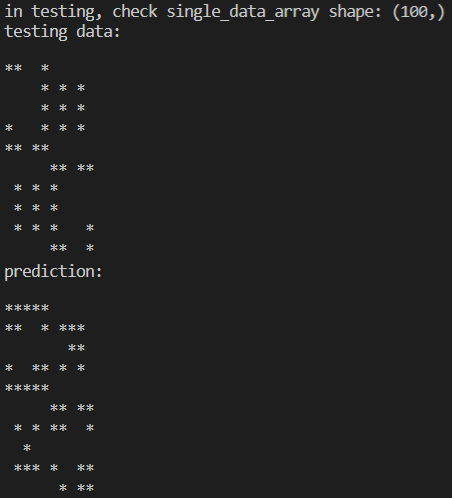
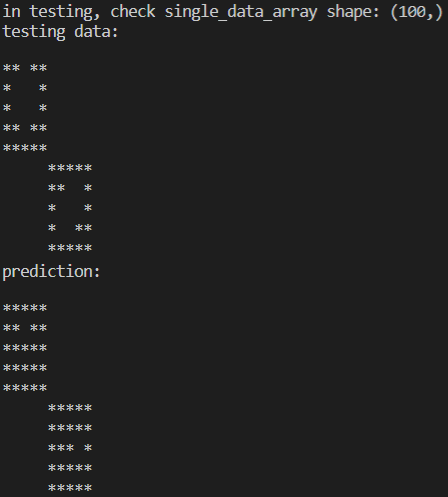
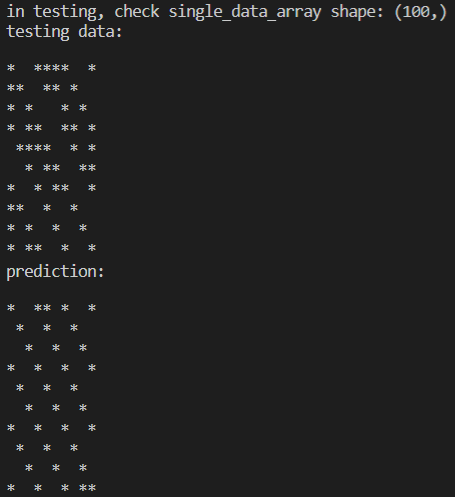
  

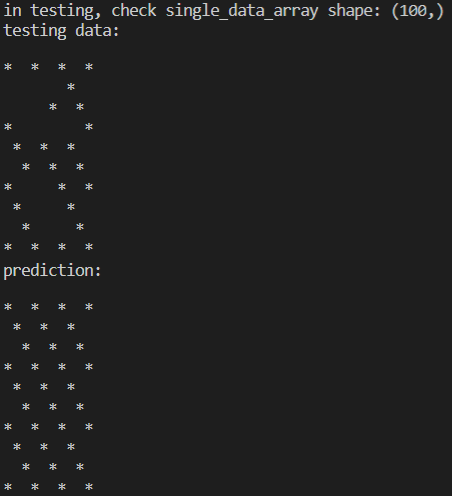
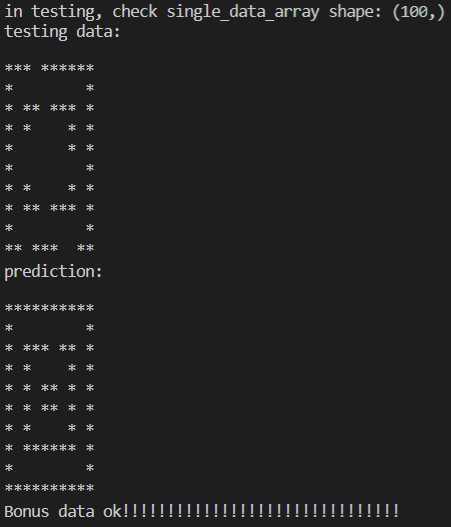
Bonus資料集：

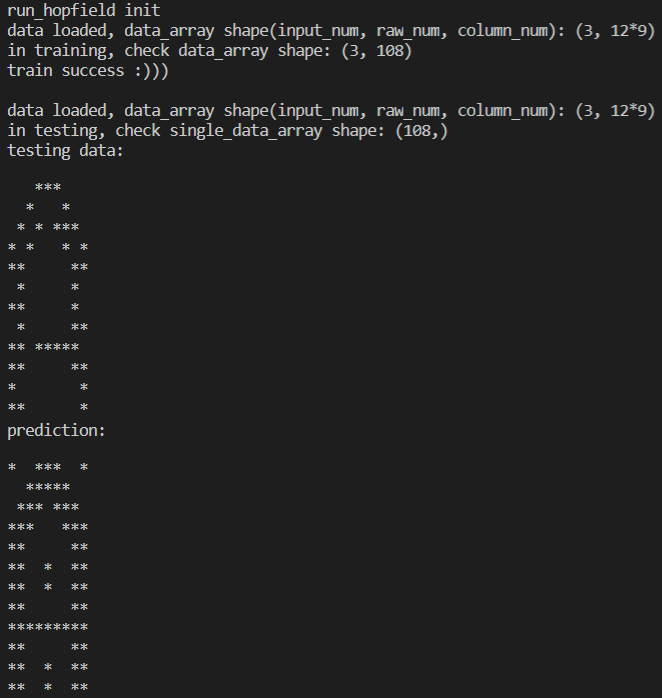
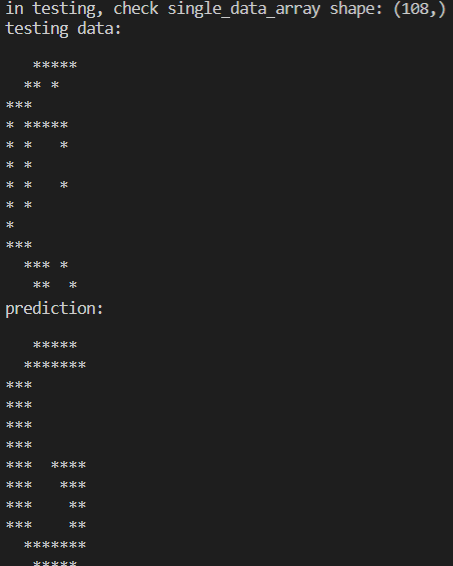
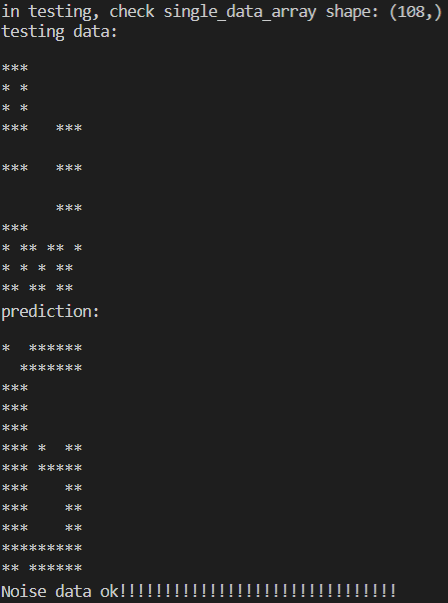
  

Noise資料集(加分題自行加入雜訊)：

1. 實驗結果分析及討論。

以上三個資料集以Basic資料集最為準確，基本上都完全回想起訓練資料，而Noise次之，Bonus最差，但是這樣直接比較並不能很好的評斷模型，畢竟輸入的矩陣大小不同請需記憶的類型也增加許多(訓練資料種類從3類變成15類)，但總的來說都有很不錯的回想能力。

主要我在權重調整上用的是Kronecker product(克羅內克積)的方法，這是兩個任意大小的矩陣間的運算，經過文獻查閱可以提升此網路的性能。

1. 如有加分項目，請在報告中說明。

加分項目：

* + 1. Bonus資料集成功
    2. 自行將訓練資料集加入雜訊，並能夠正確回想(Noise資料集)