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()) / s
ingle_data_array.shape[0]
            self.w = self.w + self.w_correction(single_data_array, sing
le_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_nu
m, 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_bon
us_noise))
    hnn = hopfield_network(training_array.shape[0], training_array.shap
e[1])
    hnn.train(training_array)

    # test and show
    raw_num, column_num, testing_array = load_data('{}_Testing.txt'.for
mat(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_bo
nus_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'
```

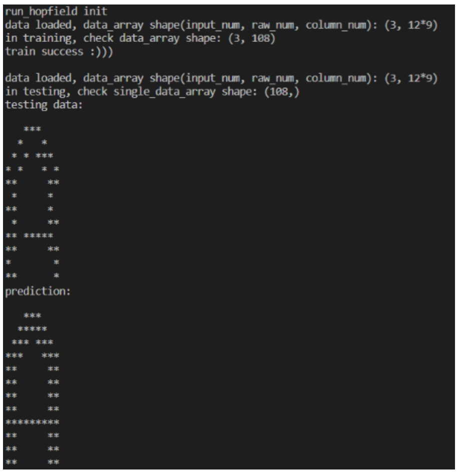
```
    if dataset_name not in legal_dataset_name:dataset_name='Basic'
    run_hopfield(basic_bonus_noise=dataset_name)

# 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()
```

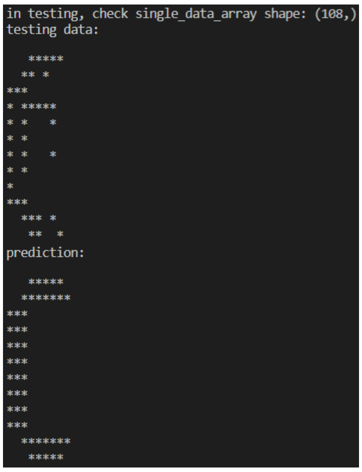
2. 程式執行說明。(如何操作、使用)

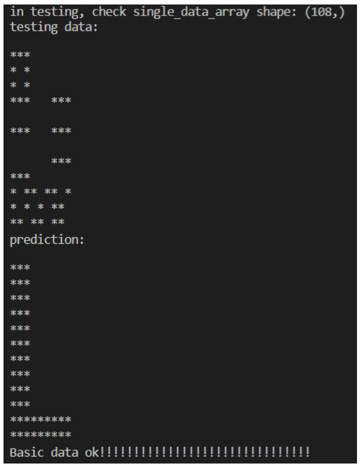
如影片所示，執行exe檔案會跑出一GUI介面，可選擇想要用以訓練及測試的資料集['Basic', 'Bonus', 'Noise']，預設及防呆都是'Basic'，即可在終端機看到精心設計的結果呈現；若直接執行.py檔案則同理，直接執行即可，一樣會有GUI介面可供應用。

3. 實驗結果(所有資料集都須有實驗結果集說明)。

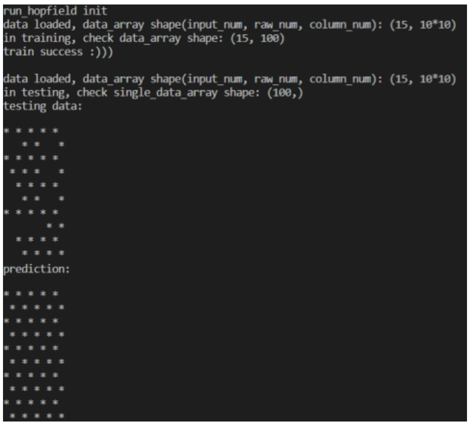
Basic資料集：

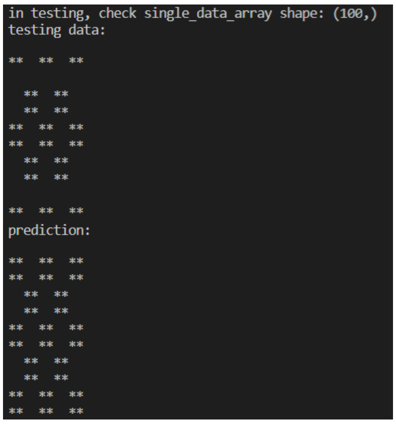


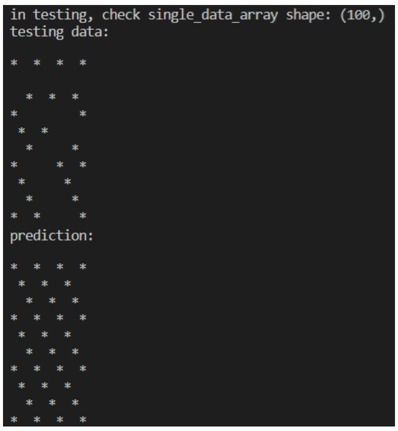


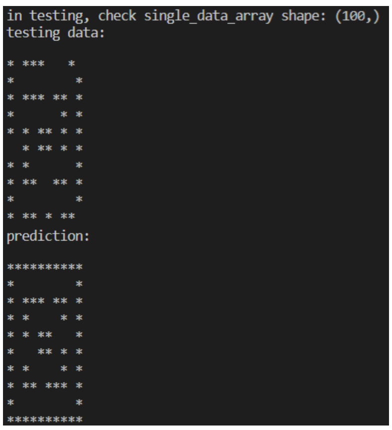


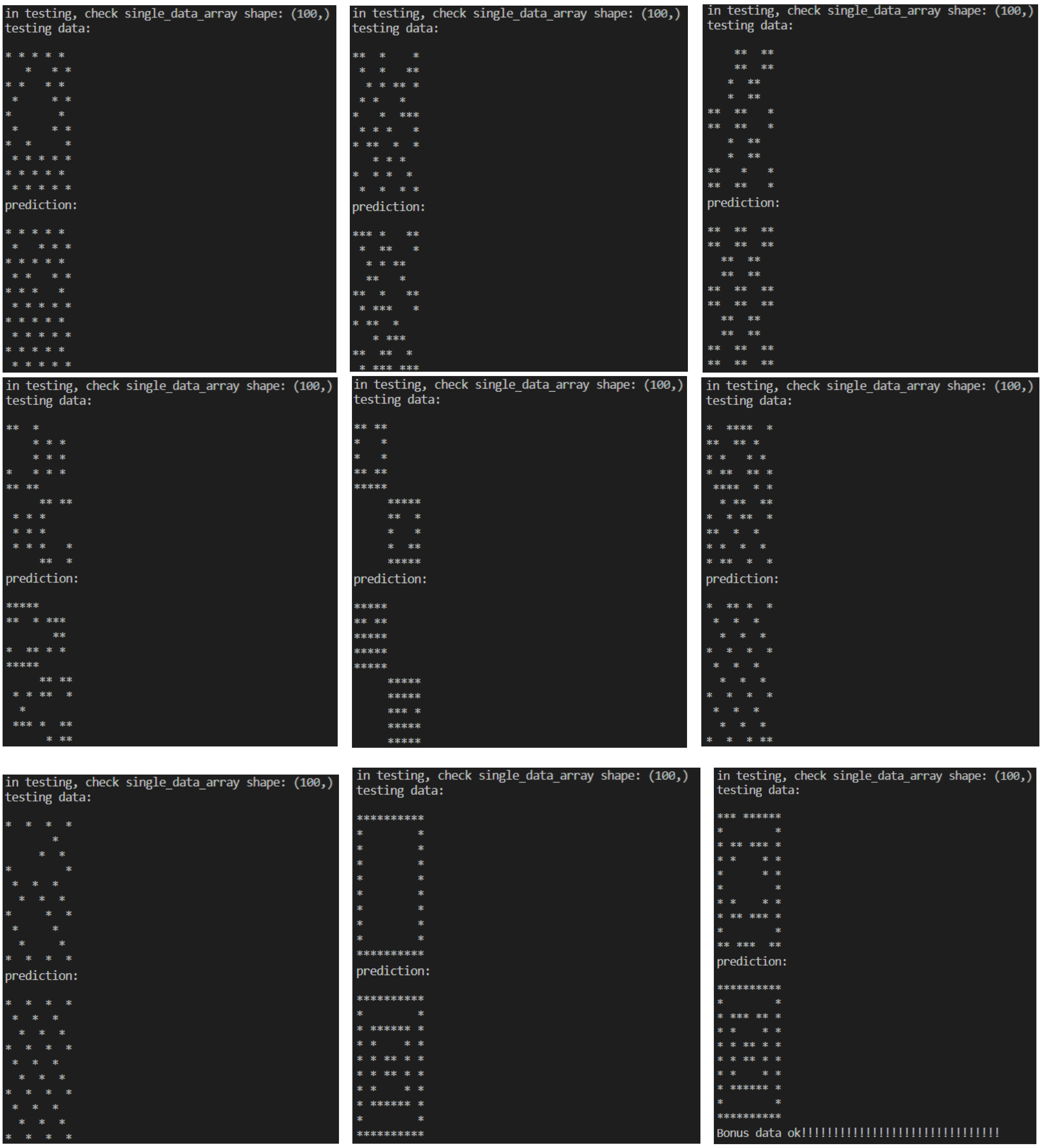
Bonus資料集：



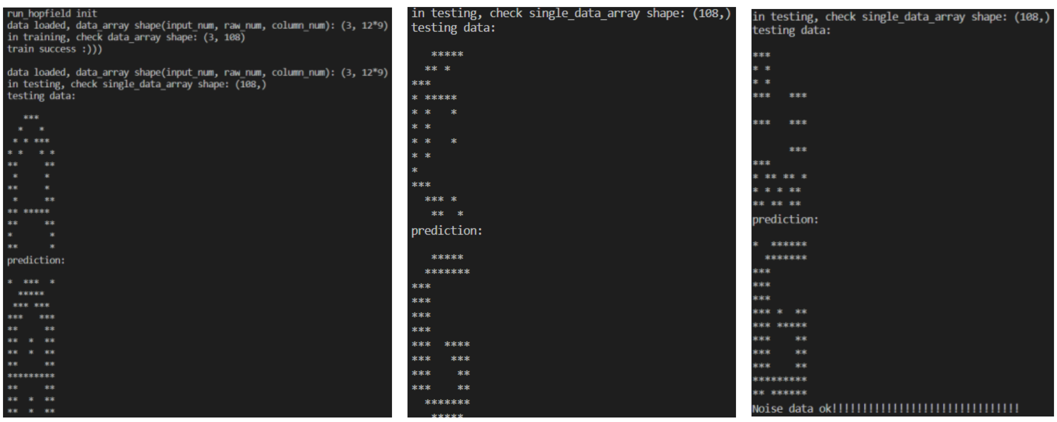








Noise資料集(加分題自行加入雜訊)：



4. 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。

5. 如有加分項目，請在報告中說明。

   加分項目：

   i. Bonus資料集成功

   ii. 自行將訓練資料集加入雜訊，並能夠正確回想(Noise資料集)