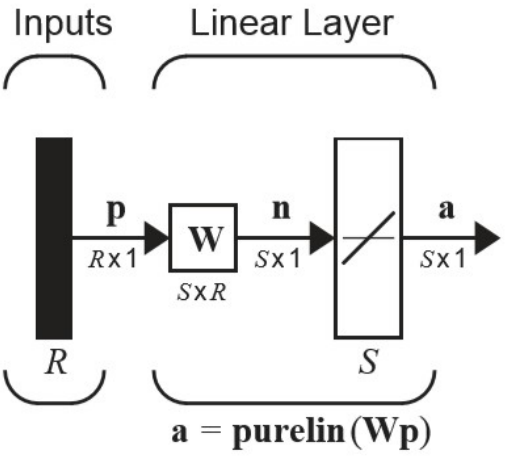
Last updated: Apr 10, 2020

類神經網路第七章應用解說

摘自本書 第七章
https://hagan.okstate.edu/NNDesign.pdf

輸入矩陣正交對Hebb Rule 的重要性

如果tranfer function 使用pruelin()的話



(1)輸入的矩陣有**滿足正交**,那麼算出來的權重不需要調整

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k. \tag{7.13}$$

The output of the network is equal to the target output. This shows that, if the input prototype vectors are orthonormal, the Hebb rule will produce the correct output for each input.

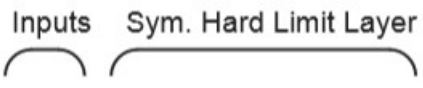
(2)輸入的矩陣**不滿足正交**,那麼算出來的權重需要透過Pseudoinverse Rule調整,
輸出(調整過後的W與輸入相乘)的誤差縮小
purelin(**Wp**) ,a=n

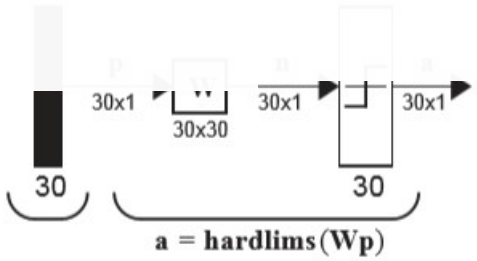
But what about non-orthogonal prototype vectors? Let's assume that each $\mathbf{p}_q$ vector is unit length, but that they are not orthogonal. Then Eq. (7.11) becomes

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k + \overbrace{\sum_{q \neq k} \mathbf{t}_q(\mathbf{p}_q^T\mathbf{p}_k)}^{Error}. \tag{7.14}$$

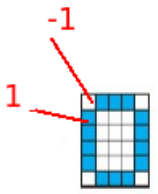
Because the vectors are not orthogonal, the network will not produce the correct output. The magnitude of the error will depend on the amount of correlation between the prototype input patterns.

但是！！如果transfer function使用hardlims()





那麼即使輸入矩陣不是正交,**算出來的權重與輸入矩陣相乘(輸出有誤差），**
也不需要調整權重 ,因為誤差的權重會佔很小不足以撼動整體輸出（後面的數字辨識會舉例）
,最後權重加總乘上輸入後,透過hardlims(**Wp**)得到輸出是1 或是-1,等於輸出會依賴所乘上權重大小判斷是否此矩陣的這個元素1有佔或是-1無佔,下圖
hardlims(**Wp**),n<0 a=-1 ,n>=0 a=1



兩個矩陣是否正交(orthongal)

假設p1 p2 符合正交那麼
p1與p2的內積=0
'表示轉置矩陣

```
>> p1 = [0.5,-0.5,0.5,-0.5]
p1 =

   0.50000  -0.50000   0.50000  -0.50000

>> p2 = [0.5,0.5,-0.5,-0.5]
p2 =

   0.50000   0.50000  -0.50000  -0.50000

>> dot(p1,p2)
ans = 0
>>
```

若p1與p2相同 內積為1,且**p1原先就是正交矩陣**,符合正交性質（orthonormal)
正交性質（矩陣是正交矩陣而且長度為1)

```
>> dot(p1,p1')
ans =  1
>> p2 = p1
p2 =

   0.50000  -0.50000   0.50000  -0.50000

>> dot(p1,p2)
ans =  1
>>
```

若p1與p2不為正交,且p1與p2不相同,內積不會等於0

```
>> p1 = [1,-1,-1]
p1 =

   1  -1  -1

>> p2 = [1,1,-1]
p2 =

   1   1  -1
```

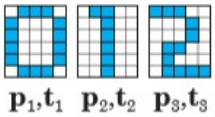
正交矩陣參考
https://www.youtube.com/watch?v=miEAqG7mc0Q

數字辨識

Application

Autoassociative Memory

Now let's see how we might use the Hebb rule on a practical, although greatly oversimplified, pattern recognition problem. For this problem we will use a special type of associative memory — the autoassociative memory. In an *autoassociative memory* the desired output vector is equal to the input vector (i.e., $\mathbf{t}_q = \mathbf{p}_q$). We will use an autoassociative memory to store a set of patterns and then to recall these patterns, even when corrupted patterns are provided as input.



$\mathbf{p}_1,\mathbf{t}_1$ $\mathbf{p}_2,\mathbf{t}_2$ $\mathbf{p}_3,\mathbf{t}_3$

The patterns we want to store are shown to the left. (Since we are designing an autoassociative memory, these patterns represent the input vectors and the targets.) They represent the digits {0, 1, 2} displayed in a 6X5 grid. We need to convert these digits to vectors, which will become the prototype patterns for our network. Each white square will be represented by a "-1", and each dark square will be represented by a "1". Then, to create the input vectors, we will scan each 6X5 grid one column at a time. For example, the first prototype pattern will be

$$\mathbf{p}_1 = \left[-1\ 1\ 1\ 1\ 1\ -1\ 1\ -1\ -1\ -1\ -1\ 1\ 1\ -1\ \ldots\ 1\ -1\right]^T. \qquad (7.41)$$

The vector $\mathbf{p}_1$ corresponds to the digit "0", $\mathbf{p}_2$ to the digit "1", and $\mathbf{p}_3$ to the digit "2". Using the Hebb rule, the weight matrix is computed

檢查 下列三項 是否符合式正交性
digit0 digit1 digit2

#the vector p1 corresponds to the digit 0
p1T = [-1,1,1,1,1,-1, 1,-1,-1,-1,-1,1, 1,-1,-1,-1,-1,1, 1,-1,-1,-1,-1,1, -1,1,1,1,1,-1]

#the vector p2 corresponds to the digit 1
p2T = [-1,-1,-1,-1,-1,-1, 1,-1,-1,-1,-1,-1, 1,1,1,1,1,1, -1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1]

#the vector p3 corresponds to the digit 2
p3T = [1,-1,-1,-1,-1,-1, 1,-1,-1,1,1,1, 1,-1,-1,1,-1,1, -1,1,1,-1,-1,1, -1,-1,-1,-1,-1,1]

# orthogonal check,**0 is ortogonal, 1 is not**
O1 = dot(p1,p2);
O1 = 0
O2 = dot(p1,p3);
O2 = -2
O3 = dot(p2,p3);
O3 = 8

表示權重算出來 乘上輸出會有誤差

$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T. \qquad (7.42)$$

(Note that $\mathbf{p}_q$ replaces $\mathbf{t}_q$ in Eq. (7.8), since this is autoassociative memory.)

Because there are only two allowable values for the elements of the prototype vectors, we will modify the linear associator so that its output elements can only take on values of "-1" or "1". We can do this by replacing the linear transfer function with a symmetrical hard limit transfer func-

我們將權重w個別算出後再個別乘上各自的輸入,如下所示

'表示轉置矩陣

權重：

W = W1 + W2 + W3

輸出：

a = WP'

a = (W1 + W2 + W3) P'

a = W1P' + W2P' +W3P'

可觀察此矩陣的某元素被權重的影響,假設P 以下圖半digit0當輸入

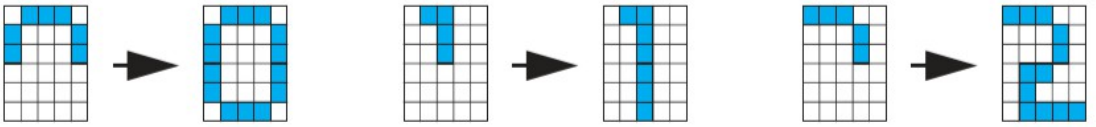


Figure 7.3 Recovery of 50% Occluded Patterns

p1T_half = [-1,1,1,-1,-1,-1, 1,-1,-1,-1,-1,-1, 1,-1,-1,-1,-1,-1, 1,-1,-1,-1,-1,-1, -1,1,1,-1,-1,-1]
p1_half = p1T_half'
a_p1_1 = W1 * p1_half
a_p1_2 = W2 * p1_half
a_p1_3 = W3 * p1_half
觀察尚未總和的輸出,會發現即使有誤差但是不大（ 此圖輸入是半digit0 ,但 a_p1_2有數值 ）
,因為最符合digit0的個別輸出 權重最大 表示影響最大

a_p1 = a_p1_1 + a_p1_2 + a_p1_3

```
>> a_p1_1'
ans =

  -16   16   16   16   16  -16   16  -16  -16  -16  -16   16   16  -16  -16  -16  -16   16   16  -16  -16  -16  -16   16  -16   16   16   16  -16

>> a_p1_2'
ans =

  -10  -10  -10  -10  -10  -10   10  -10  -10  -10  -10  -10   10   10   10   10   10   10  -10  -10  -10  -10  -10  -10  -10  -10  -10  -10

>> a_p1_3'
ans =

    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

a

```
>> a_p1T = a_p1'

a_p1T =

  -26    6    6    6    6  -26   26  -26  -26  -26  -26    6   26   -6   -6   -6   -6   26    6  -26  -26  -26  -26    6  -26    6    6    6    6  -26
```

將a帶入hardlims (a = WP) 即可得到輸入p半digit0 等於 訓練資料的digit0
y_p1 = hardlims(a_p1T);

```
>> y_p1 = hardlims(a_p1T);

>> check_is_digit0(y_p1);

p1T =

  -1   1   1   1   1  -1   1  -1  -1  -1  -1   1   1  -1  -1  -1  -1   1   1  -1  -1  -1  -1   1  -1   1   1   1   1  -1

this input value is digit 0
>>
```

訓練資料的digit0
p1T = [-1,1,1,1,1,-1, 1,-1,-1,-1,-1,1, 1,-1,-1,-1,-1,1, 1,-1,-1,-1,-1,1, -1,1,1,1,1,-1]

以上程式碼由 digits_recognition.m 執行