COMP390

2020/21

A teaching app to demonstrate Hebb's learning rule

Student Name: Jiahui.Li

Student ID: 201447830

Supervisor Name: Alajlan,Saad

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Acknowledgement

Here I would to like to express the thanks to my supervisors Biktasheva, irina and Alajlan, Saad. Their help both in the area of neural network and advice on academic writing contribute to the accomplishment of this project in a great way.

UNIVERSITY OF
LIVERPOOL

COMP 390

2020/21

A teaching app to demonstrate Hebb's learning rule

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Abstract

The hebb's rule is an important concept in the area of neuroscience. However, due to the lack of learning application, the only way to learn this knowledge is listening to tutorial and reading books. To supplement the learning experience and stimulate the interest of student, this project is put forward. The object of this project is to illustrate the process of hebb's rule with image recognition and to show the training and applying process with Qt language. The application allows user to customize their input image and set up different parameters to see the purpose of parameters. The final product works steadily and the feedback from users indicates the satisfied experience for them.

Contents

List of Figures

1.Introduction & Background

In this chapter, I am going to give some introduction about Hebb's rule, Qt creator and waterfall model to help reader understand the project. Hebb's rule is core algorithm of the project, Qt creator is the developing platform, and waterfall model is the model to manage the project

1.1 Hebb's rule and normalization

Hebb's rule is known as an neural network learning rule since it was introduced by Donald Hebb[1]. It explains that learning pattern of the neuron network by adjusting the neuron synapse. From then on, Hebb's rule is also applied in the process of training unsupervised neural network.

Hebb's rule is based on Hebb's assumption: an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell.

From the perspective of artificial neuron network, it can be explained as if a positive input a_i^k leads to a positive output X_j^k , the weight w_{ji}^{k+1} should be increased. There is a mathematic explanation for it, i.e.

$$w_{ji}^{k+1} = w_{ji}^k + \Delta w_{ji}^k \quad (1)$$

Where $\Delta w_{ji}^k = C a_i^k X_j^k$, a_i^k is i_{th} element of network input in the k_{th} iteration, X_j^k is the j_{th} element of network output in the k_{th} iteration, C is the learning rate which is predefined. From this equation, it can be found that weight will increase not only in the occasion of both input and output are positive but also in the occasion that they are both negative. The weight only decreases if the input and the output have different signs.

It can also be noticed that there is no terminating condition for the equation which means that the learning process will iterative itself endlessly and weight will increase or decrease without limitation.

To solve this drawback, Hebb's rule can be augmented by normalization rule. With normalization rule, a neuron's predisposition will be sharpened and make its firing better correlated with stimulus pattern. normalized hebb's has the similar mathematic equation to original hebb's rule, which is, i.e.

$$w_{ji}^{k+1} = \frac{w_{ji}^k}{\sqrt{\sum {w_{ji}^k}^2}} + \Delta w_{ji}^k \tag{2}$$

Where $\Delta w_{ji}^k = C a_i^k X_j^k$,Where $\Delta w_{ji}^k = C a_i^k X_j^k$, a_i^k is i_{th} element of network input in the k_{th} iteration, X_j^k is the j_{th} element of network output in the k_{th} iteration, C is the learning rate which is predefined

By introducing the $\sqrt{\sum {w_{ji}^k}^2}$ as the denominator, the total strength of weights can be kept to a constant. Additionally, a terminating condition is set up, i.e.

$$\max(w_{ji}^k - w_{ji}^{k-1}) < \mathrm{Q} \tag{3}$$

Where Q is the threshold of the error which is predefined.

This condition means if the difference between any old weight and its updated weight is less than the threshold, the learning process terminate.

Normalization and terminating condition help hebb's rule perform better in the application such as unsupervised learning or self-organization network.

1.2 Qt creator

Qt creator is the cross-platform development environment. By integrating the code editor and Qt designer, it allows user to build and edit the interface directly, which decrease the amount of coding and simplify the development process. With its low-level APIs of the platform it supports, the Qt creator achieves its flexibility and efficiency, see Fig.1.

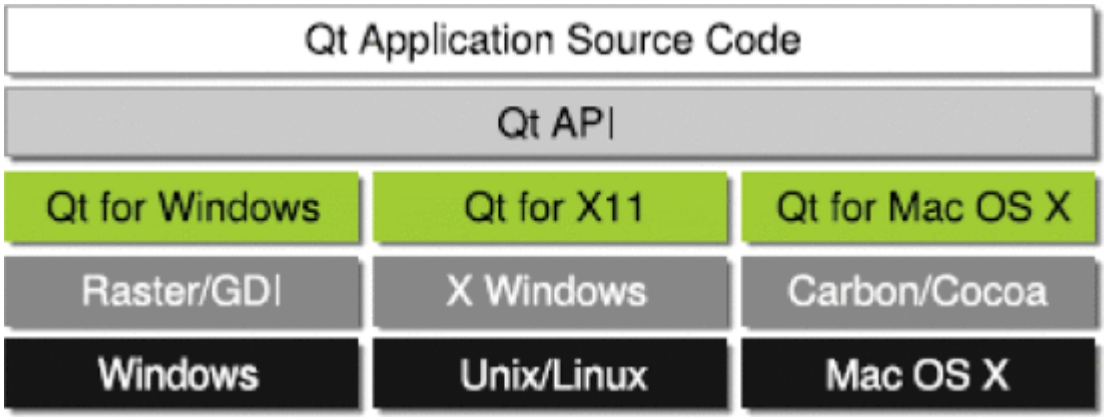

Figure 1. overview of Qt's architecture on supported desktop platform [2]

A Qt project consists of headers, sources, forms and resources, see Fig.2.
headers and sources contain the code of the project. In Forms, the user can directly see the interface and edit the interface using different widgets instead of writing codes. Resources is used to contained the file aside from the project to be used, such as imported picture.
In Qt creator, there are some stand-alone classes can facilitate the design of interface. The QGraphicsTextItem class provides a text Item to display formatted text. The QGraphicsLineItem class provides a line item to draw a line. QGraphicsPixmapItem provides a pixmap item which display the image. The QGraphicsEllipseItem provides an ellipse item [4]. The QGraphicsRectItem provides a rectangle item. The QGraphicScene provides a surface for managing a large number of 2D graphical items.
There is a useful feature called signal and slots in Qt which are used for the communication between objects [5]. Signals are emitted by an object when its internal state has changed which might influence object's client or owner. A slot is called when a signal connect to it is emitted.

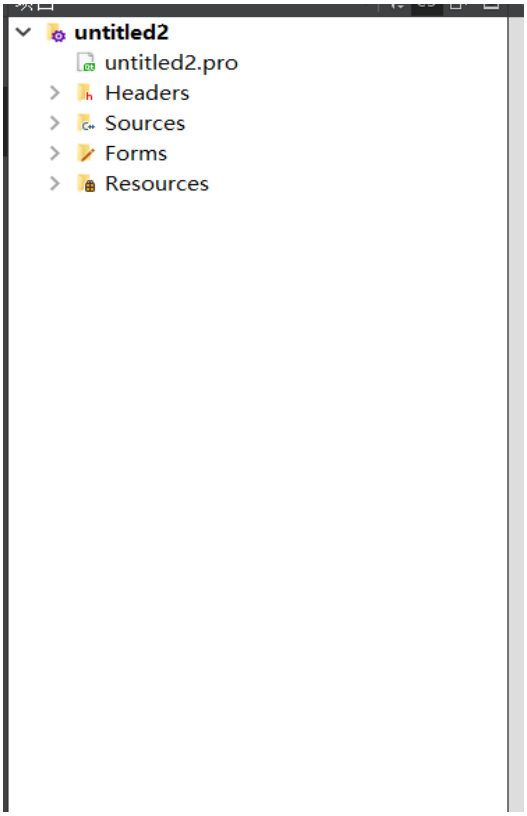

Figure 2. Structure of qt project

1.3 Waterfall Model

Waterfall model is the approach to break the project activities into linear phases, including requirements, design, implementation, verification, maintenance. Every phase comes after a phase which is completed and the output of one phase becomes input of next phase [3]. Waterfall model is a formal and ineffective model, any change in early phase will lead to rework of the later phase.

1.4 Association

Association is the task of mapping pattern to pattern. For instance, if there is an incomplete or corrupted pattern, using association can lead to the response of a stored pattern which corresponds to the input pattern, i.e.

$$S = a * w^T = \sum_{i=1}^{n} w_i a_i \quad (4)$$

Where a is the input pattern, w is the stored pattern or weight in term of networks.

This equation will generate several output (S_1,….S_j), only max(S_j) is considered as the desired output and it also implies w_j is the most similar pattern to a_j.

1.5 Objectives

- Constructing an app that illustrates the different stage of hebb's rule to users
- Verifying the correctness of hebb's rule using the app
- Introducing roles of different parameters in hebb's rule

1.6 Literature review

Hebb's has been wildly applicated in different regions based on his inner reflection to the nature.

S. Akio and Y. Masaka proposed a new CPG(central pattern generator) model and Hebbian learning rule for the CPG to understand the principals of animals' motion control. The main role of Hebbian rule is to generate the desired motion in their project [6].

I. Ahmad, P.P. Mondal and R. Kanhirodan also apply Hebbian learning in their algorithm which exploits the inter pixel correction of image and then performs optimal smoothening of the noisy images. It successfully decreases the Normalized Mean Square Error in their test [7].

It can be observed Hebbian rule has been a strong support tool in different areas of the science. However, there is little materials about introducing Hebbian rule itself. The reason behind this is Hebbian is primary learning rule which is assumed to be learned by scientists. However, for those who never approached unsupervised learning, Hebbian rule might be hard to understand.

Therefore, there is extreme demand for an application which can help those who has never learned hebb's rule to understand it in an efficient way.

2. Design

In this chapter, the organization of the system, the data structure, the design of user interface and the algorithm are presented. By understanding the algorithm and the data structure, user can enhance their knowledge about hebb's rule. And by knowing the design of user interface and data structure, user can learn how to develop the program which facilitates their own project development.

2.1 System organization

The system organization consists of a major interface, which is linked to 3 sub interfaces and 3 sub interfaces which can be divided further, see Fig.3. The major user interface can be seen by users once they open the application. Interfaces of Train with original Hebb's rule, train with normalized Hebb's rule and test are sub interfaces of the major one, each of them is linked by a corresponding push button in the major interface. In each sub interface, the user can choose the function of selecting the picture, setting the parameter or checking out the internal calculation of training or testing.

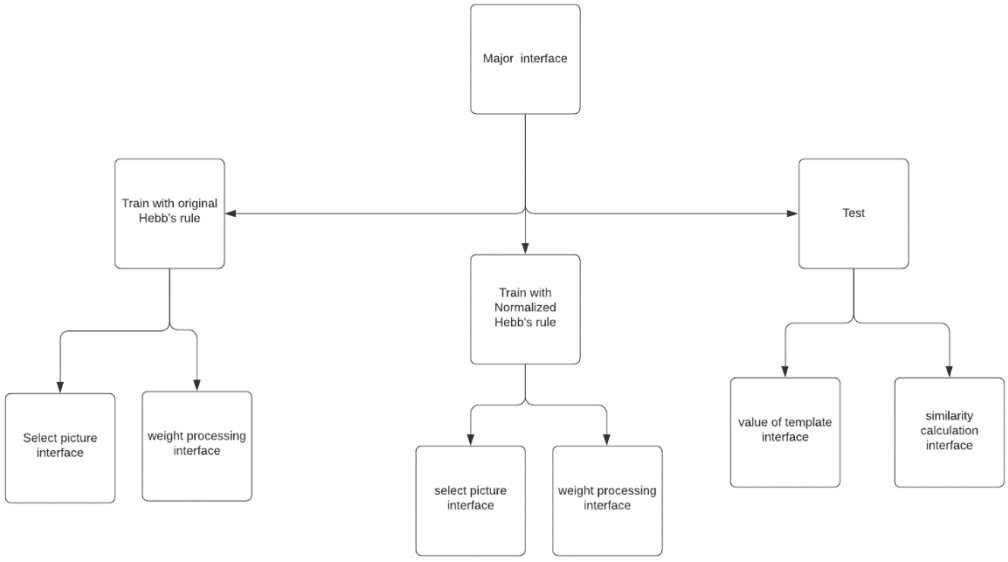

Figure 3. the interface organization

2.2 data structure and algorithm

The data structure in this project refers to the data type of container to contain the input, temporary and output data. And algorithm refers to Hebb's rule, Normalization rule and association rule, this part is presented in the form of pseudo code.

2.2.1 data structure

In the function of Train with Hebb's rule, the input data is the int array with size of 30, of which the element can only be 1(indicate the colour of black) or -1(indicate the colour of white), see Figure.4. The weight data is the double array with size of 30. The class QGraphicsTextItem, QGraphicsLineItem are used to draw the text and line respectively. The threshold is an int value and learning rate is a double value.

In the interface of Train with Normalized Hebb's rule, the input data, see Fig.4, and weight data are stored in the int array, and the double array respectively. The text is drawn by the QGraphicsTextItem, the line is drawn by QGraphicsLineItem. The threshold is an int value and learning rate is a double value.

In the function of Test. The input is stored in the int array with size of 30, of which the element can only be 1(indicate the colour of black) or -1(indicate the colour of white). The requirement for data of templates is as same as the one for input data. The template of number 0 to 9 are provided, see Fig.5 to Fig.14. The output image is stored in QGraphicsPixmapItem. The similarity is presented by an integer.

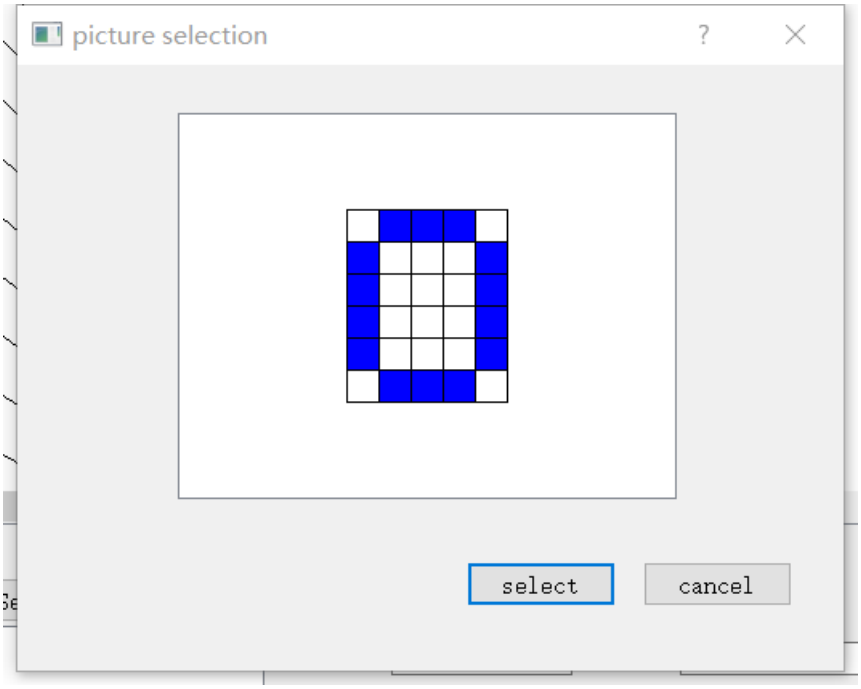

Figure 4. Example of input data

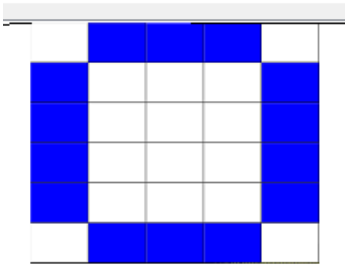

Figure 5. Template of number 0

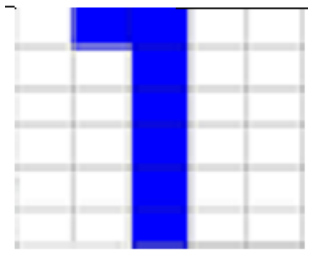

Figure 6. Template of number 1

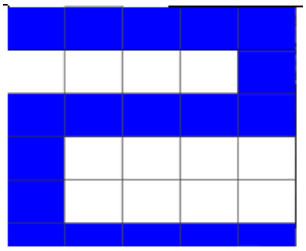

Figure 7. Template of number 2

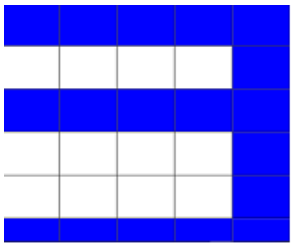

Figure 8. Template of number 3

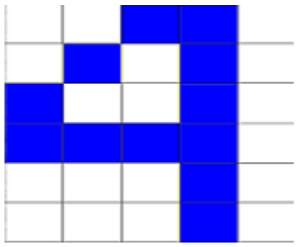

Figure 9. Template of number 4

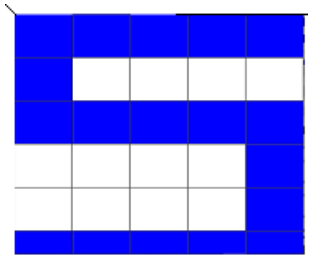

Figure 10. Template of number 5

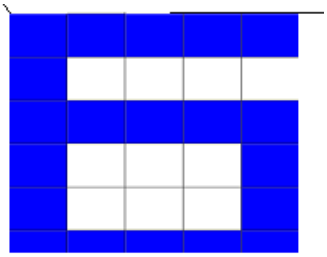

Figure 11. Template of number 6

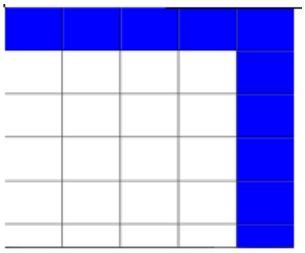

Figure 12. Template of number 7

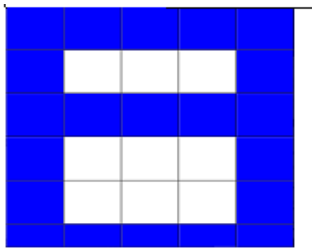

Figure 13. Template of number 8

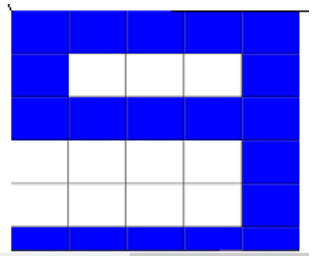

Figure 14. Template of number 9

2.2.2 Algorithm

The algorithm applied in the interface of Train with Hebb's rule is the original Hebb's rule. The original Hebb's rule in the form of pseudocode is given.

```
Sum<-0
Value<-c
X<-0          // initial all parameters
Get t
DO for i=0 to size{
        product <- a[i]* weight[i]
        sum <- sum+product
}       // sum the multiplication of input and
weight
If sum is larger or equal to t then{
        X <- 1
}       // determine to output
QVector<qreal> dealt_weight <-0
DO for i=0 to size{
        dealt_weight[i]<-a[i]*C*X
        weight[i] <- weight[i]+dealt_weight[i]

}               //update weight
```

(1)

The algorithm applied in the function of Train with normalized Hebb's rule is normalized Hebb's rule. The normalized Hebb's rule in the form of pseudocode is given.

```
QVector<double>  old_weight<-0, new_weight<-0
max_delta<-0
continue <-1
Get size
Get t
Get C // initial all parameters
DO for i=0 to size{
        old_weight[i]<-weight[i]
}//store the old weight
Mod<-0
DO for i=0 to size{
        mod <-mod+ weight[i]*weight[i]
}// calculate the mold of weight
mod<-sqrt(mod)
DO for i=0 to size{
        weight[i] <- weight[i]/mod
}       //calculate the weight divided by mold
sum <- 0
X <- 0
DO for i=0 to size{
        sum <- sum+ value of weight[i]
}
If sum is larger than t then{
        X <-1
}
QVector<double> dealt_weight<-0
DO for i=0 to size{
        dealt_weight[i]<- C*a[i]*X
        weight[i]<-weight[i]+dealt_weight[i]
}       // update the weight
DO for i=0 to size{
        Append weight[i] to the back of new_weight
}
DO for i=0 to size{
        If I new_weight[i]-old_weight[i] I is bigger than max_delta then{
                max_delta <- I new_weight[i]-old_weight[i] I
        }
}//calculate difference between old weight and new weight
If max delta is less than t then{
        continue <- 0
}
```

(2)

The algorithm applied in the function of test is association rule. association rule in pseudocode is given

```
total <-0
Get weight_a
Get weight_b
size <- 30// initial all parameter
DO for i=0 to size{
        total <-total+weight_a[i]*weight_b[i]

}       //calculate the total
```

(3)

2.2.3 Design of User interface

The major interface includes three buttons, which are the Train with original Hebb's rule, The Train with normalized Hebb's rule and the Test, see Fig.15.

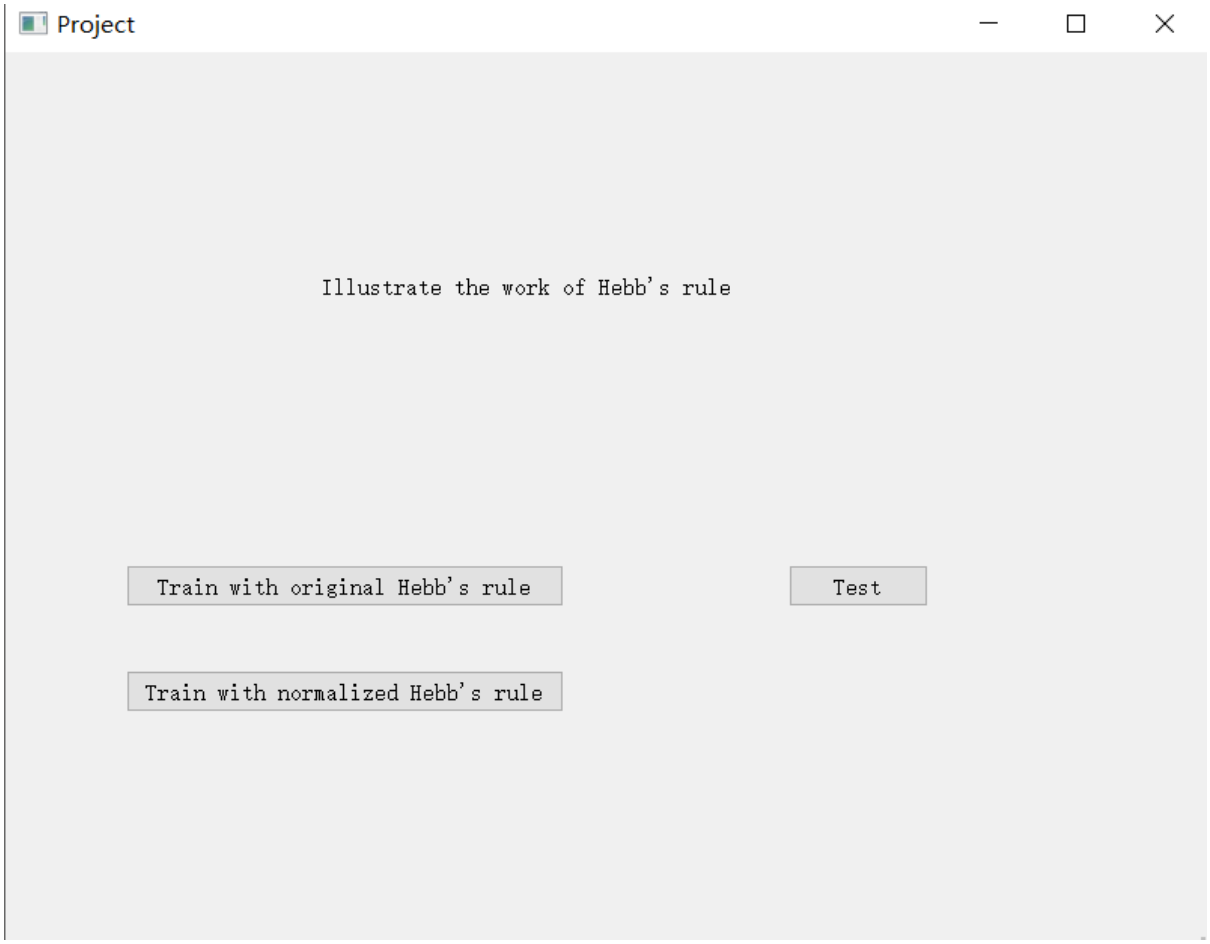

Figure 15. major interface

The interface of Train with original Hebb's rule consist of 4 graphic boxes, 2 input boxes and 3 buttons, see Fig.16. The illustration box is on the top-left of the interface, which occupy the largest space. This box is used to illustrate the data of weight and input. The input box and weight box are on the top-right, they hold the table of weight data and input data. The learning rate box and threshold box allow users to set different parameters to observe the influence of these parameters.

The button of select picture links to the interface of picture selection, see Fig.17. by clicking the cube, the colour of cube will turn into blue and the internal data will be changed between 1 and 0.

The box of weight links to the interface of weight calculation, see Fig.18. It shows the detail of weight calculation.

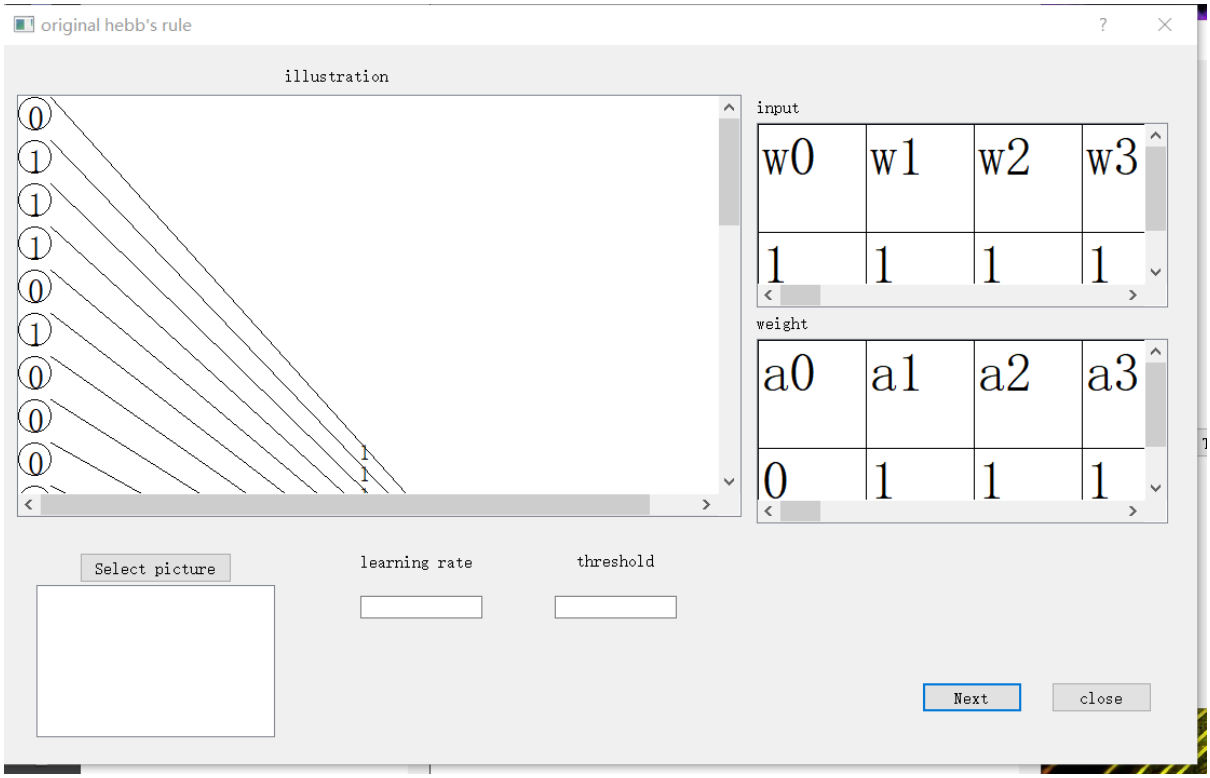

Figure 16. interface of Train with original Hebb's rule

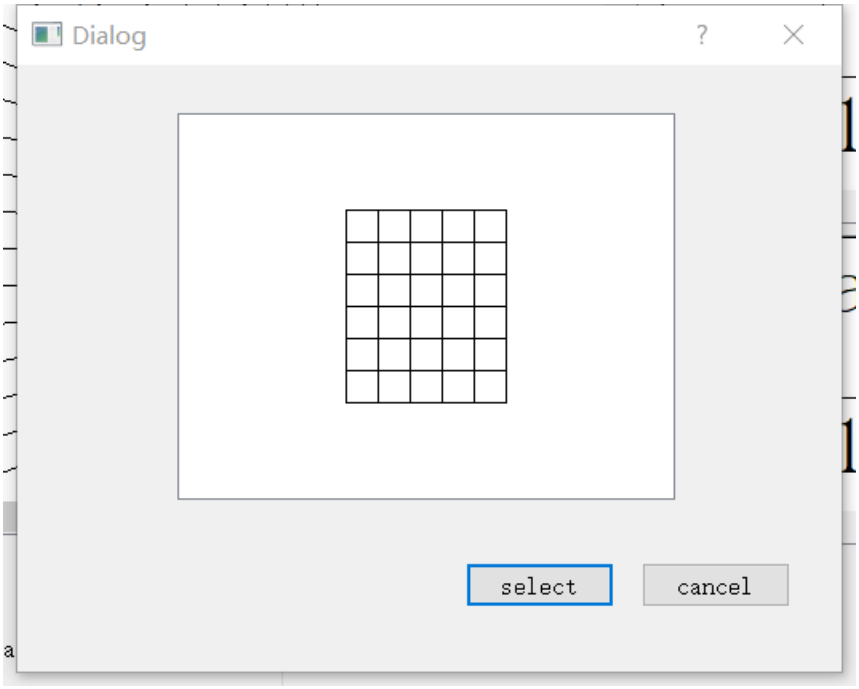

Figure 17: interface of selecting picture

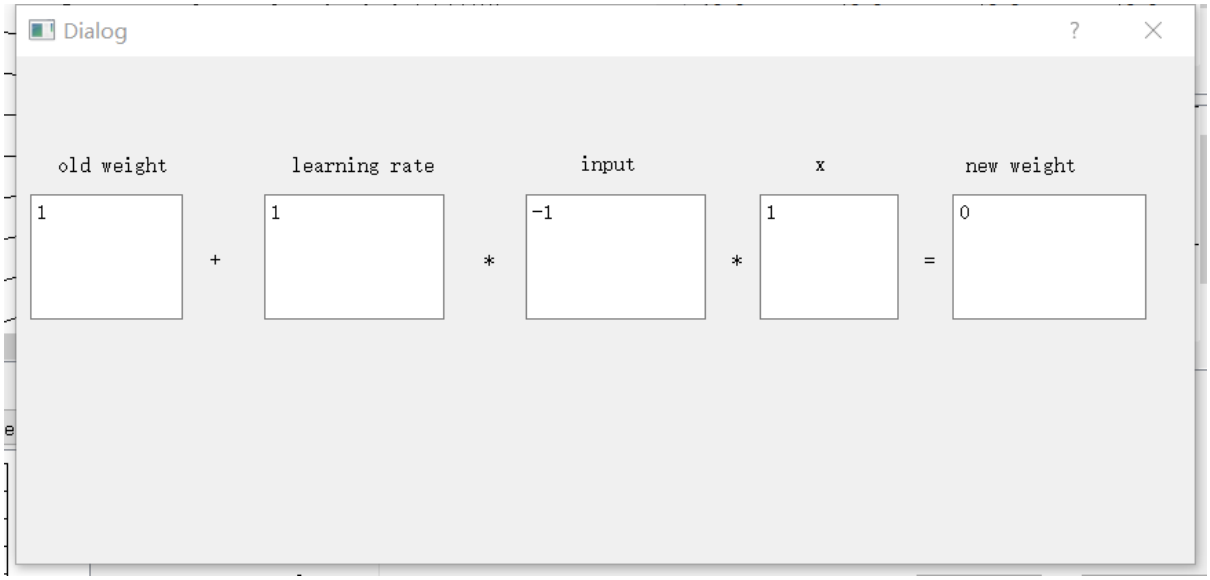

Figure 18: interface to illustrate the process of weight calculation

The interface of Train with normalized Hebb's rule is shown in Fig.19. In this interface, the illustration box illustrates the input data and weights, the input box shows the current value of the weight and the weight box records the current value of the input. The picture selection interface is applied in this interface, see Fig.6. The difference box shows the amplitude of each update, if this value is below the threshold, the program will terminate. The weight box is clickable and shows the interface of weight calculation, see Figure 20. The threshold and learning rate box help user to set up different values for this parameter.

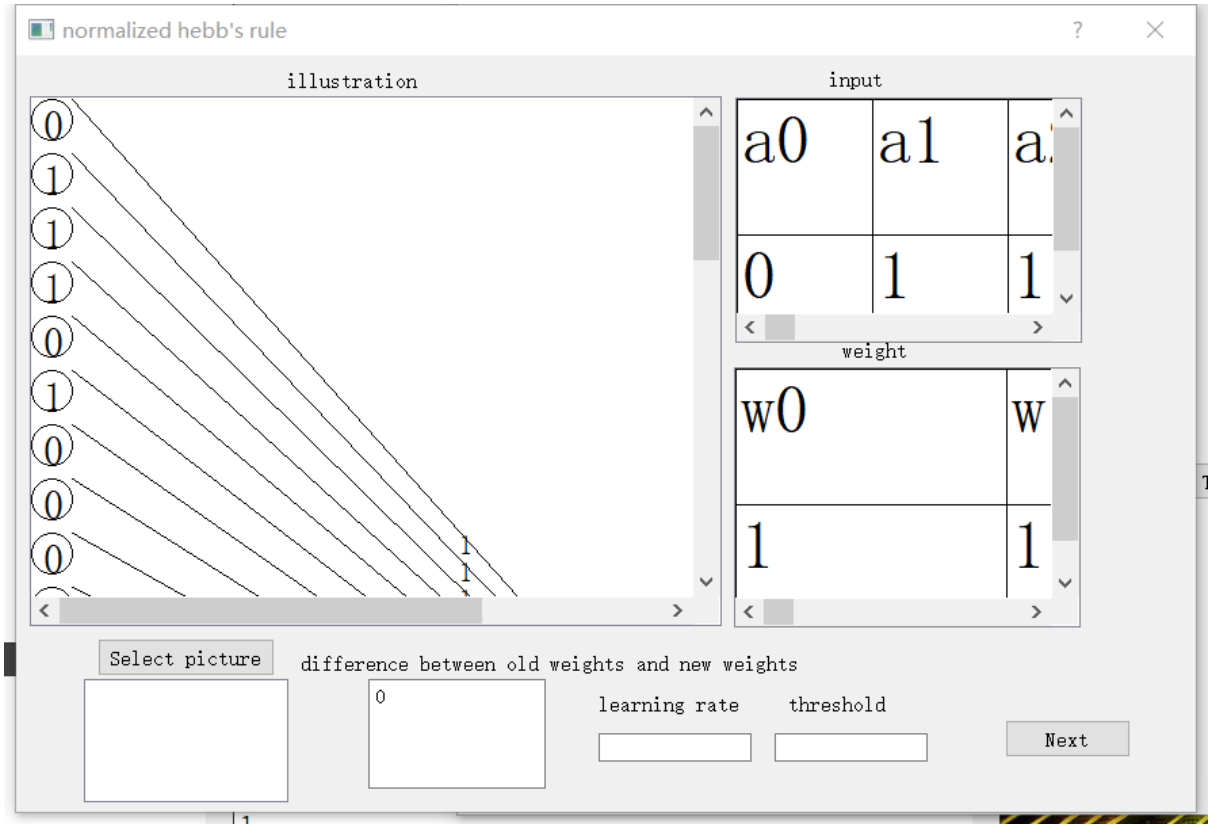

Figure 19. interface of Train with normalized Hebb's rule

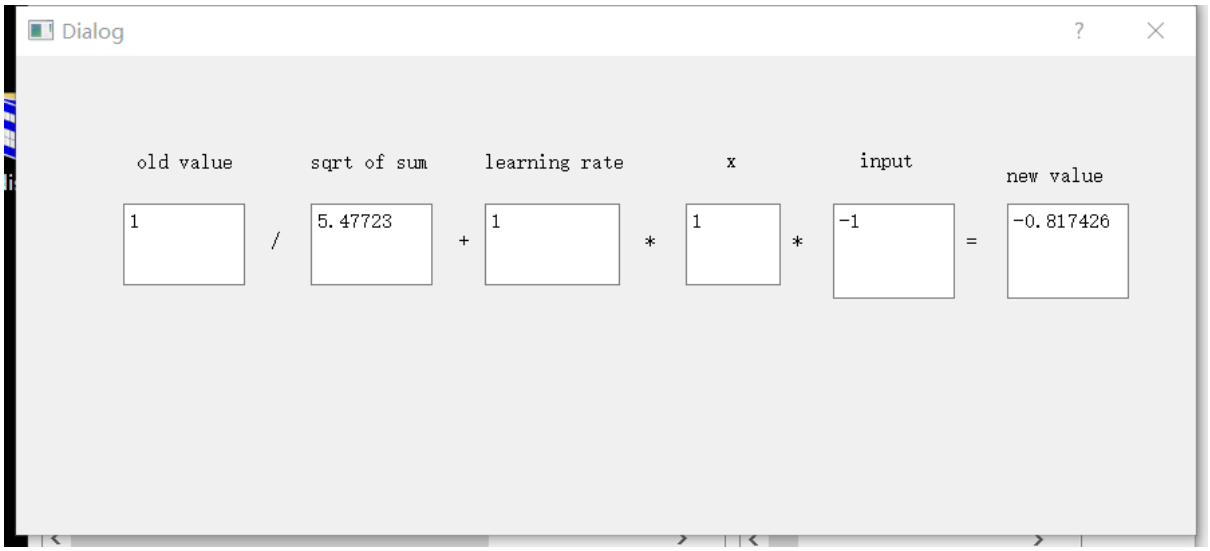

Figure 20. interface of weight calculation

The interface of Test is shown in Fig.21. In this interface, the illustration box illustrates the input image, the template images and the value of association. The output box shows the value of association. The best fit box gives the most similar template. Picture selection button links to interface which is used to select the picture, see Fig.17. The element in

illustration box is clickable, which link to the interface of the value of template and the interface of association calculation, see Fig.22 and Fig.23.

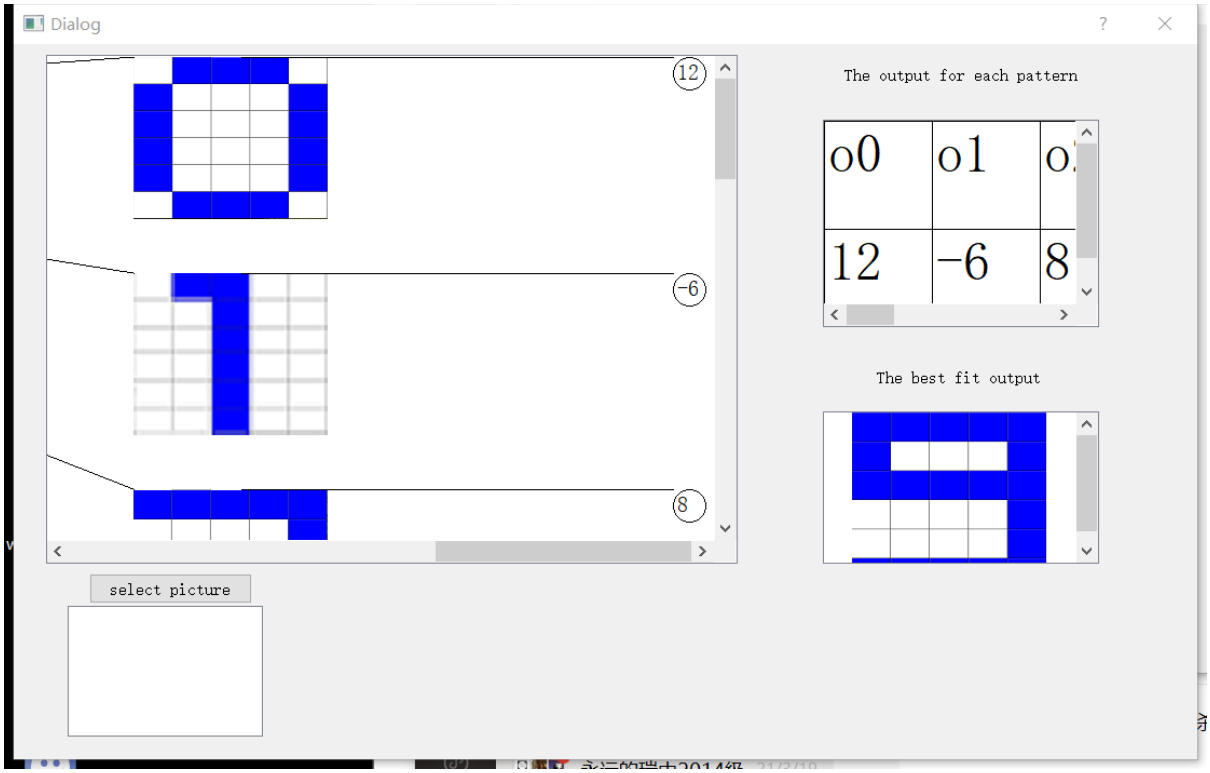

Figure 21. the interface of Test

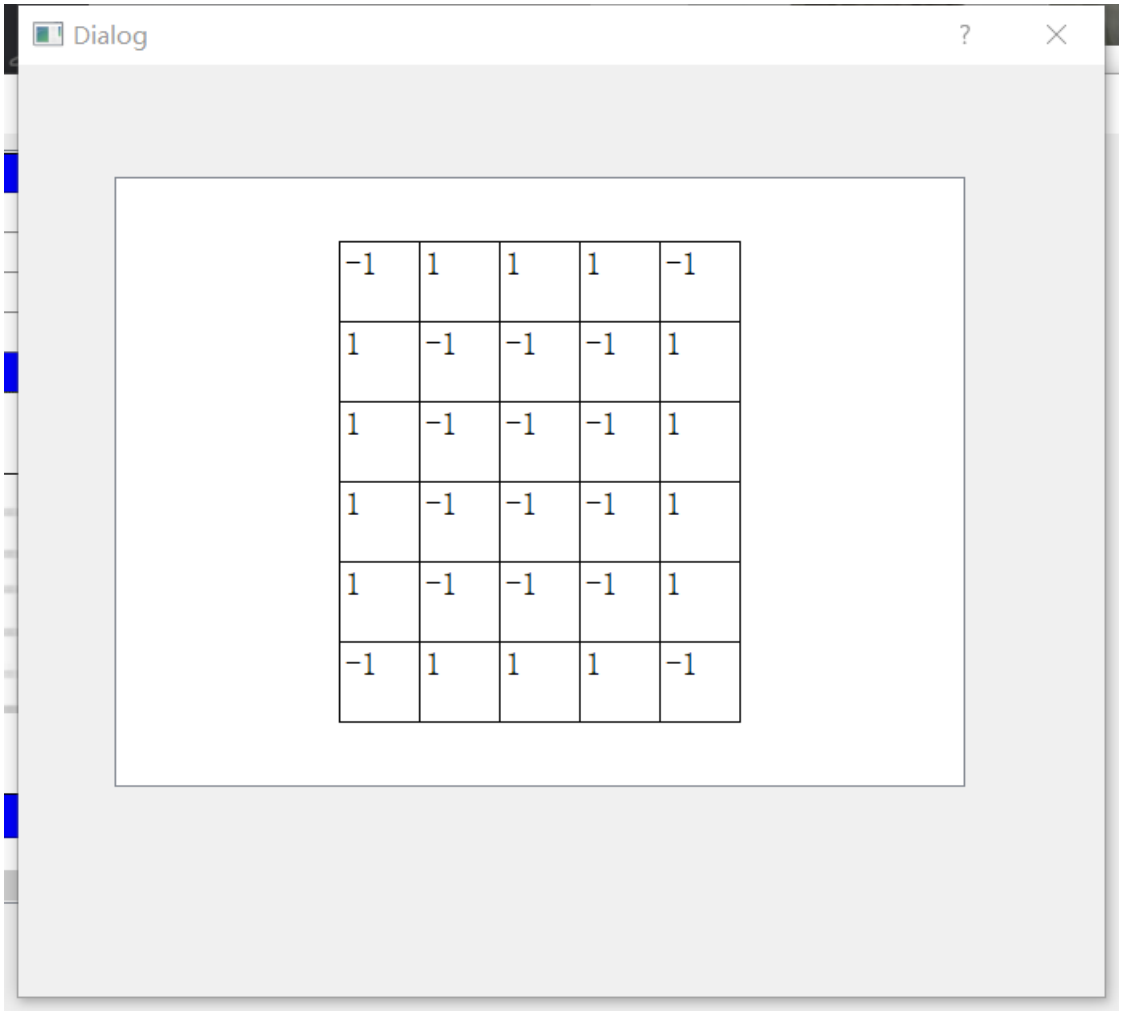

Figure 22. The interface of value of template

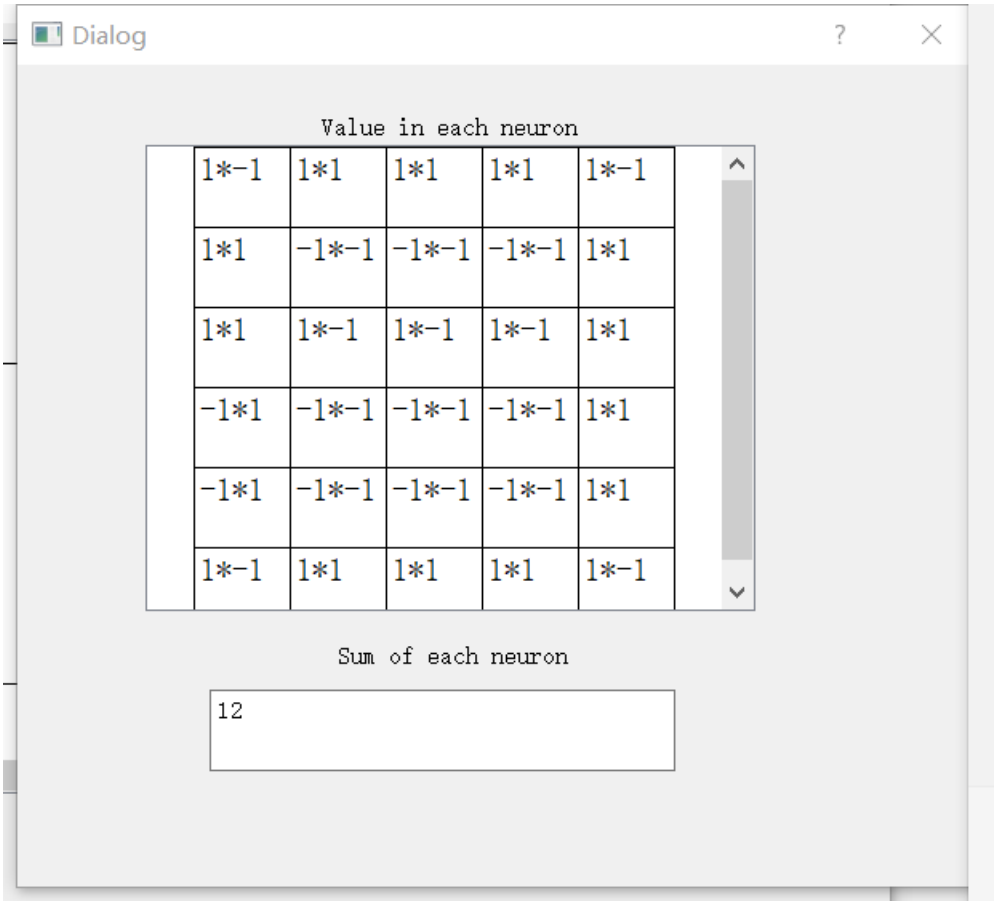

Figure 23. The interface of association calculation

2.2.4 The flow of navigation

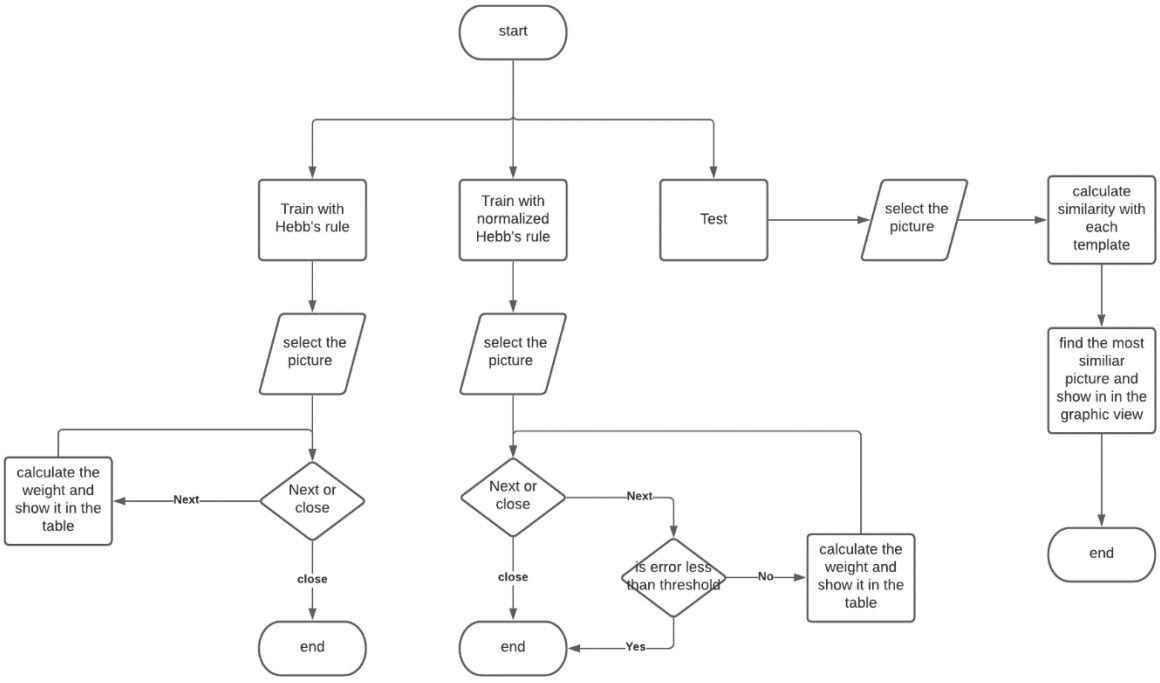

Figure 24: the flow control between interfaces

The flow of program includes three major interfaces train with hebb's rule, train with normalized hebb's rule and test, see Fig.24. The branch of train with hebb's rule links to interfaces of picture selection, weight calculation of original hebb's rule. The branch of train with normalized hebb's rule links

to interfaces of picture selection, weight calculation of normalized hebb's rule. The branch of test links to interfaces of picture selection, association calculation and template data.

3 Implementation

In the implementation chapter, the detail of development is presented, covering from the stage of planning to implementing the project with codes. This chapter will helper user to learn the skill of managing the project within limited time and the usage of C++ in qt platform.

3.1 planning

At the beginning of project, only the purpose of project is given: A teaching app to demonstrate the Hebb's rule. However, there is no restriction to the method to explain it. After realizing the popularity of unsupervised learning in the area of image recognition, it is decided to use image recognition to illustrate the Hebb's rule.

Then it needs to be decided to follow which kind of model to implement the project. Considering the simplicity of waterfall model, this model is adopted through the project. Following each stage of waterfall model, a time schedule is generated, see Fig.25. The stages include designing, coding, getting review, writing report and preparing the introduction video within the time of 51 days.

	App to illustrate Hebb rule	**51 days?**	**Mon 01/02/21**	**Mon 12/04/2**
	Designing	**10 days**	**Mon 01/02/21**	**Fri 12/02/21**
	Deciding the topic	3 days	Mon 01/02/21	Wed 03/02/21
	deciding the language	2 days	Thu 04/02/21	Fri 05/02/21
	interface designing	5 days	Mon 08/02/21	Fri 12/02/21
	coding	**21 days?**	**Mon 15/02/21**	**Mon 15/03/21**
	implementing	6 days	Mon 15/02/21	Mon 22/02/21
	testing and Debugging	15 days	Tue 23/02/21	Mon 15/03/21
	getting review	4 days	Tue 16/03/21	Fri 19/03/21
	writing report	5 days	Mon 22/03/21	Fri 26/03/21
	preparing the introduction video	8 days	Thu 01/04/21	Mon 12/04/21

Figure 25. The time schedule of the project

Before starting the coding, choosing a suitable language is also critical. Because of Qt is taught in the module of ELEC362, it is considered the most efficient and suitable language.

3.2 coding

The critical part of programming is to reproduce the process of hebb's learning using programming language. Therefore, the equation of Hebb's rule needs to be converted into the form of code. First step is to convert the original Hebb's rule. The equation form of original Hebb's rule is expressed by Eq.1

During the process of coding this part, one important problem is to determine the data structure. At first, all parameters are set as int. Then the problem arisen, the weight will automatically round up if it includes decimal. And learning rate needs to be decimal in the reality. Therefore, parameters C and weight are changed to double.

In the original Hebb's rule, the amount of update depends on threshold, input and learning rate. As long as sum of input multiple weight is larger than threshold, each of weight will change by the amount of input multiple learning rate.

In normalized Hebb's rule, there is two additional step compared to the original Hebb's rule. The first step is normalization, before the update, the weights need to be normalized.

Each weight is divided by the sqrt of sum of square of each weight. The second step is calculating the error, after the update, calculating the biggest error between the new weight and old weight, if this value is bigger than the threshold, continue the update, otherwise stop the update.

One problem met in this part is the reservation of number of decimal places. Because there are too many numbers of decimal places of weight to be hold in limited space of box. It is decided to only keep 3 decimal places. The way to cut down decimal places is transforming the weight into the form of string with keeping three decimals and transform it back to double.

In the calculation of association, the weight is multiplied by the input and the multiplication is the association which represent the similarity between input and template.

Explaining the process of hebb's rule is also an important part of the application. Therefore, it is designed when the user clicks the template or value of association, there is interface that pop off to show the user what the value of this image is or how this weight is calculated. To enable the click response, new classes is built for them.

The class of myEllipseItem inherited from QGraphicsEllipseItem, which allow the user to add the ellipse item to the QGraphicsScene. Superclass of the inheritance depends on the data type needs to be stored.

The function of clicking is achieved by reimplementing the function of mousePressEvent, which allow the program to response to the press action of the mouse. In this example, the function of mousePressEvent generate an interface of calculation when it is called.

The clickable template is the class of MygraphicsItem which inherited from QGraphicsPixmapItem, because stored data is the image instead of number. the mousePressEvent create an interface of test data to show the internal value of template.

The clickable rectangle inherited from QGraphicRectItem. By clicking each rectangle, it will change its colour to blue and set the flag to 1 or change it inversely.

Another important part is about communication between interfaces. There are features called signal and slot help to implement the communication.

The signal of class Selectpicture() called senddata() send out the data in the form of int. And class Test has a slot called receive_data() to accept that value. The function Connects() combine the signal and slot to enable the data to be transmitted successfully.

4. Testing and evaluation

In this chapter, unit testing and evaluation will be presented. Unit testing test the availability of each unit, including functions and interfaces to make sure each element can finish its work individually. Evaluation test the collaboration between units, it regards elements as a whole project and invite testers to user the project and give the feedback. Evaluation can reflect the overall quality of the project.

4.1 Unit testing

Unit testing reflect the availability of each unit in the project. Each unit need to be tested individually before the user test. This make sure each unit work properly.

4.1.1 Image selection

This test is conducted to verify that the program allow user to select the picture and accept the input data correctly. At the beginning, elements of input are all white, see Fig.17. And it's original data is listed in Table.1. It can be observed the default value of element is set to be -1. This is for the facility of association calculation.
Then to generate the example input, several elements are clicked, see Fig.26. It can be observed after clicking the element, the colour is turned into blue. And the value is changed into 1 where the element is clicked, see Table.2.

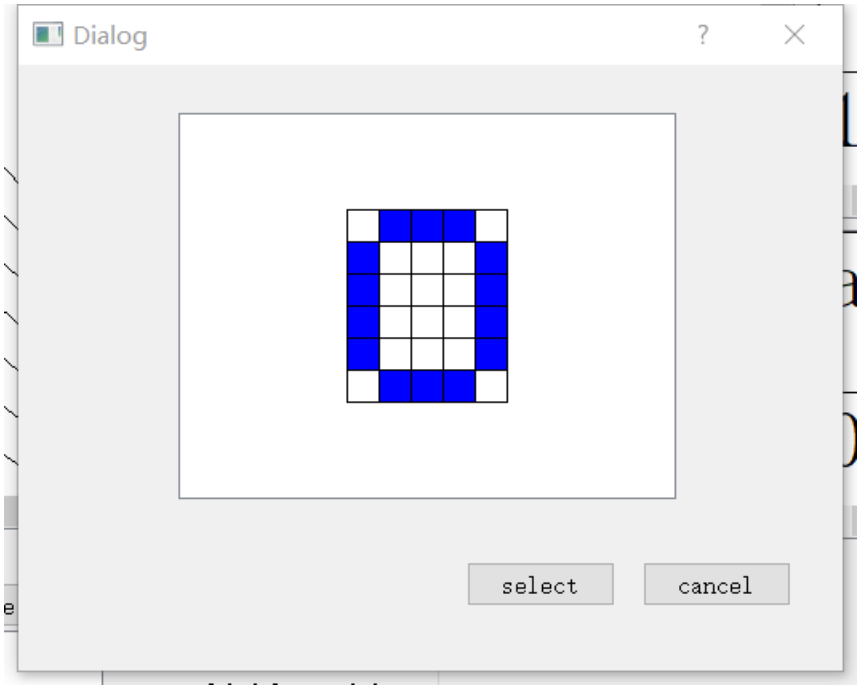

Figure 26: The test input

Table 1. The data of original input

-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

Table 2: The data of test input

-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

4.1.2 Signal transmission

To verify the success of signal transmission, the test input in Fig.26 is used again. At the beginning, there is no image in the box of select picture, see Fig.23. Then, since the select button in Fig.26 is clicked, the image has been appeared in the box, see Fig.27. By comparing it with the test input selected. It can be concluded that the signal transmission is successful.

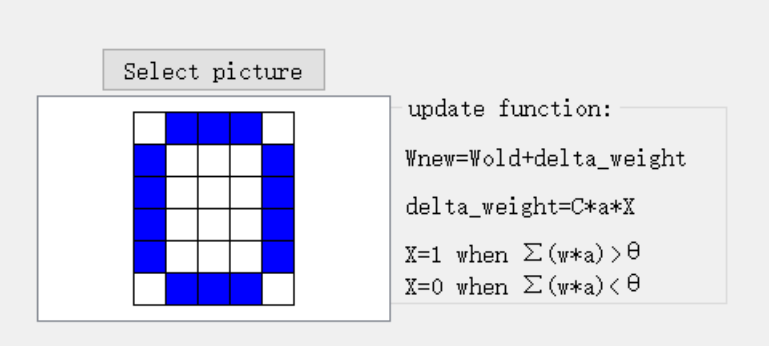

Figure 27: The received image

4.1.3 The original hebb's rule

After selecting the picture, next step is to verify the training process. Using test data in Fig.26 as the input, then setting the threshold and learning rate to -30 and 1 respectively, the result of first iteration is shown in Fig.28.

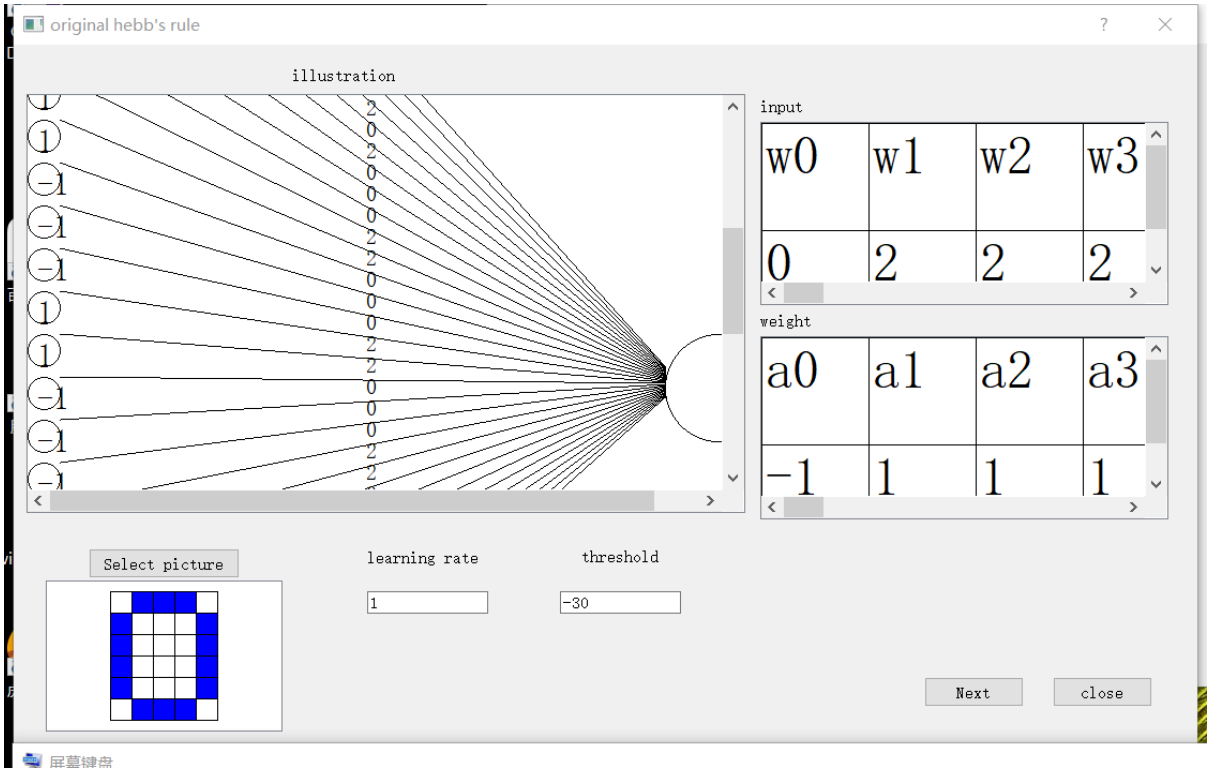

Figure 28: the result of first iteration of the original hebb's rule

Then recording the result of first neuron in first 10 iterations, the result is shown in Table.3

Table.3 result of original hebb's rule with threshold=-30, learning rate=1

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	0	-1	-2	-3	-4	-5	-6	-7	-8	-9

Then keeping the threshold same, changing the learning rate to 0.1 and 10, recording the result of first neuron in first 10 iteration, the result is shown in Table.4 and Table.5

Table.4 result of original hebb's rule with threshold=-30, learning rate=0.1

Iteration time	1	2	3	4	5	6	7	8	9	10

The value of weight	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0

Table.5 result of original hebb's rule with threshold=-30, learning rate=10

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	-9	-19	-29	-39	-49	-59	-69	-79	-89	-99

Comparing the result in Table.3, Table.4 and Table.5. It can be observed that the amount of each change is -1,-0.1,-10 respectively which are proportional to the learning rate. And it can also be observed the amount of change is same for the same condition of learning rate and threshold.

Then keeping the learning rate to 1, changing the threshold to -10 and 0. The result of first neuron in first 10 iteration is recorded in Table.6 and Table.7.

Table.6 result of original hebb's rule with threshold=-10, learning rate=0.1

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	0	-1	-2	-3	-4	-5	-6	-7	-8	-9

Table.7 result of original hebb's rule with threshold=0, learning rate=0.1

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	1	1	1	1	1	1	1	1	1	1

Comparing Table.3, Table.6 and Table.7. It can be observed the threshold decides whether the weight will change or not. If the threshold is above certain value, the weight will stop updating.

Then the internal process of weight calculation is verified, see Fig.29. it can be observed the calculation result is 0, as same as the first result shown in Table.3.

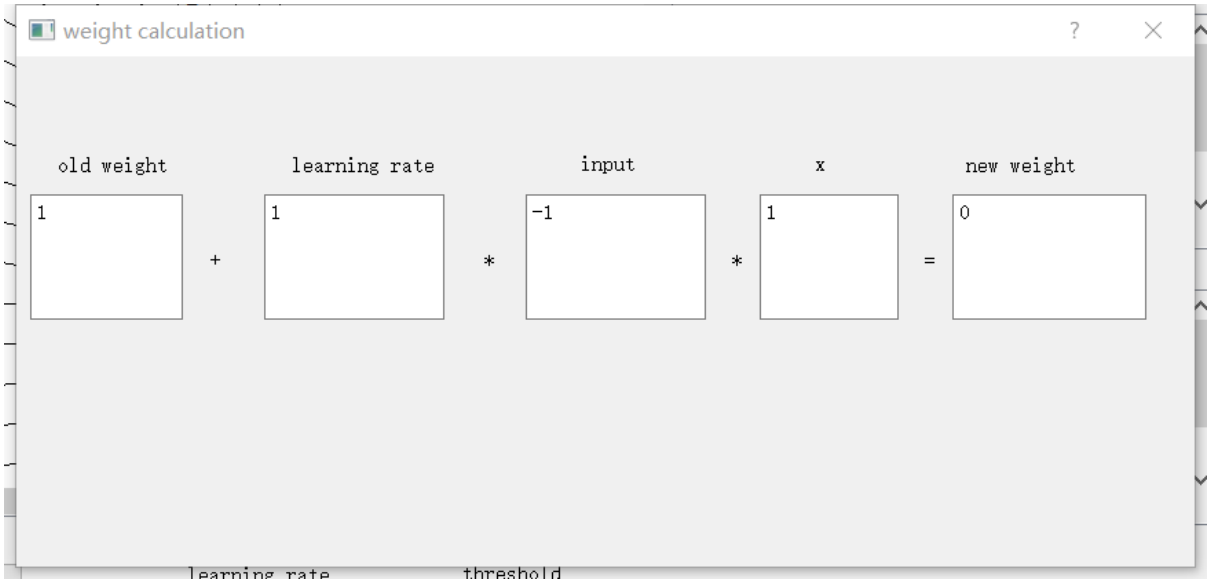

Figure 29: The internal process of weight calculation

4.1.4 Normalized hebb's rule

To test the working state of normalized hebb's rule, the test data in Fig.33 is used as input. In the first test, the threshold and learning rate is set to 0 and 1 respectively, the result of first iteration is shown in Fig.30.

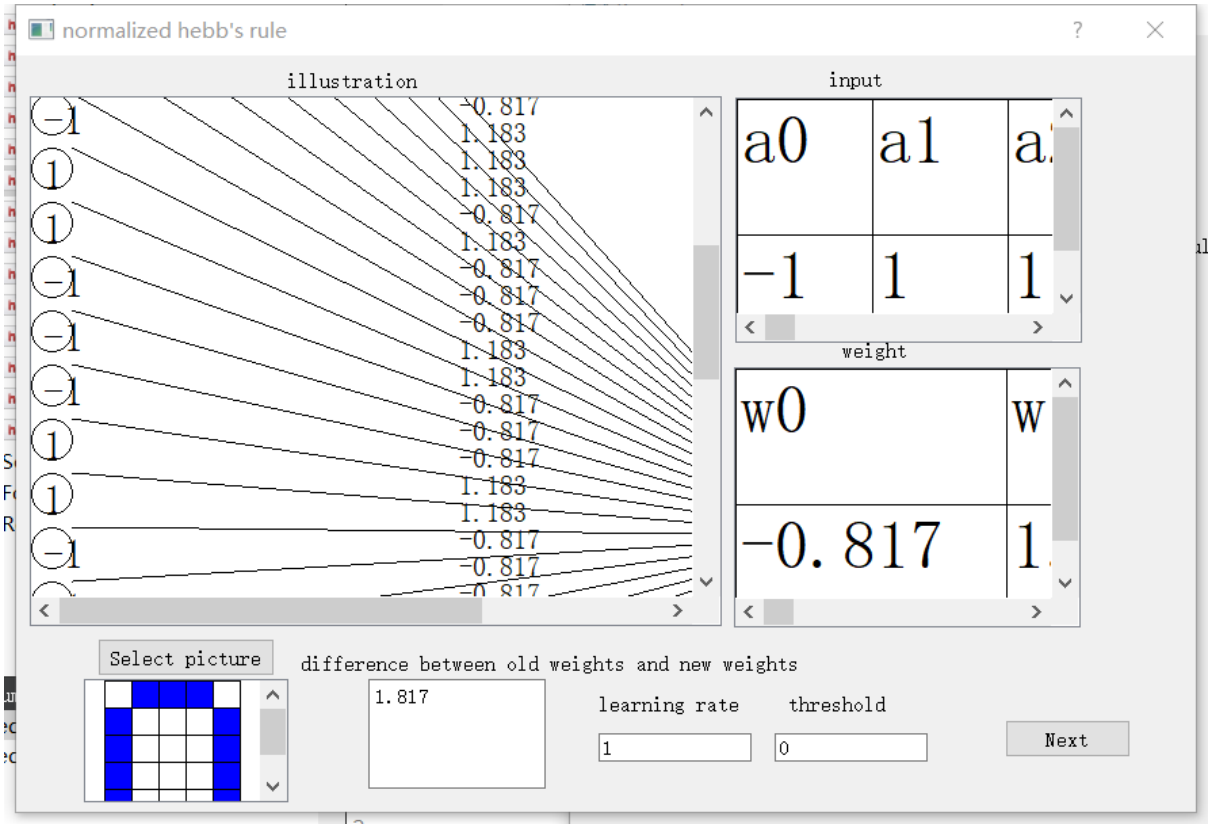

Figure 30. The first iteration of Normalized Hebb's rule

Then recording the result of first neuron in ten iterations, see Table.8

Table8. result of normalized hebb's rule with threshold=0,learning rate=1, theta=0.01

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	-0.817	-1.148	-0.178	/	/	/	/	/	/	/

From Table.8, it can be observed the trend of result is not unidirectional, the reason behind this is guessed to be in the third iteration, the sum of the weight multiple the input is less than threshold, therefore the update rule is not applied to the weight while the normalization is still applied. And only first three iteration is applicable, because in the last seven iteration, the largest difference between new weight and old weight is less than theta, the program terminates.

Then setting the threshold to 10, keeping learning rate at 1, theta at 0.01, the result can be seen in Table.9

Table 9. result of normalized hebb's rule with threshold=10,learning rate=1, theta=0.01

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	0.183	/	/	/	/	/	/	/	/	/

There is only one iteration available in this test. It means the only normalization rule is applied to the weight once, the update rule is not applied to the weight at all. After the first iteration, the difference between new weight and old weight is 0, therefore the program terminates.

Then setting the threshold to -10, keeping learning rate at 1, theta at 0.01, the result can be seen in Table.10.

Table 10. result of normalized hebb's rule with threshold=-10,learning rate=1, theta=0.01

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	-0.817	-1.148	-1.178	-1.182	/	/	/	/	/	/

From Table.10. It can be observed the trend of change is unidirectional. It implies in each iteration, normalization rule is applied as same as updated rule until the difference between the new weight and old weight is less than the theta.

Then setting the threshold to -10, learning rate to 0.1, keeping theta at 0.01, the result can be seen in Table.11.

Table 11. result of normalized hebb's rule with threshold=-10,learning rate=0.1, theta=0.01

Iteration time	1	2	3	4	5	6	7	8	9	10

The value of weight	0.083	-0.025	-0.119	-0.182	-0.221	-0.244	-0.259	-0.267	/	/

From Table.11, it can be observed the rate of change is slower with decreasing of learning rate and iteration time increase due to this.

Then setting the threshold to -10, learning rate to 10, keeping theta at 0.01, the result can be seen in Table.12.

Table 12. result of normalized hebb's rule with threshold=-10,learning rate=10, theta=0.01

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	-9.817	-10.179	-10.183	/	/	/	/	/	/	/

From Table.12. It can be observed in the first iteration, the weight drops very fast, from 1 to -9.817, then becomes stable and terminate in third iteration. It turns out the learning rate control the speed of first iteration, but have limited influence to the subsequent iterations.

Then setting the threshold to -10, learning rate to 0.1, theta to 0.001, the result can be seen in Table.13

Table 13. result of normalized hebb's rule with threshold=-10,learning rate=0.1, theta=0.001

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	0.083	-0.025	-0.119	-0.182	-0.221	-0.244	-0.259	-0.267	-0.273	-0.276

Comparing Table.13 with Table.11, it can be observed values in first 8 iteration are totally same, the difference is at the last three iteration, program terminate with theta=0.01, but continue with theta=0.001.

To make it clearer, setting the threshold to -10, learning rate to 0.1, theta to 0.1. The result is shown in Table.14.

Table 14. result of normalized hebb's rule with threshold=-10,learning rate=0.1, theta=0.001

Iteration time	1	2	3	4	5	6	7	8	9	10
The value of weight	0.083	-0.025	-0.119	/	/	/	/	/	/	/

It can be observed with these conditions, only 3 iterations are successfully executed and in first three iteration, values are as same as those with different theta. Therefore, it can be concluded that the theta influences the number of iteration but not influence the result in each iteration.

From the test, different roles of parameters are discovered. The threshold decides the number of updates, but it doesn't influence the normalizations. While, on the contrast, the theta influences the normalization but doesn't affect the updates. The learning decides the size of each update. It should be better to set those parameters to a practical range to make result more reasonable.

4.1.5 The test result

To test the function of test, using the data in Fig.26 as the input. The result of application is shown in Fig.31. The output of each template has been listed in Table.15. The internal data of template 0 is given, see Fig.32. And the calculation of output 0 is shown in Fig.33. From Table.15, it can be observed that the largest output is the output with template 0 and the smallest output is the output with template 1. Therefore, the most similar template to input is 0.

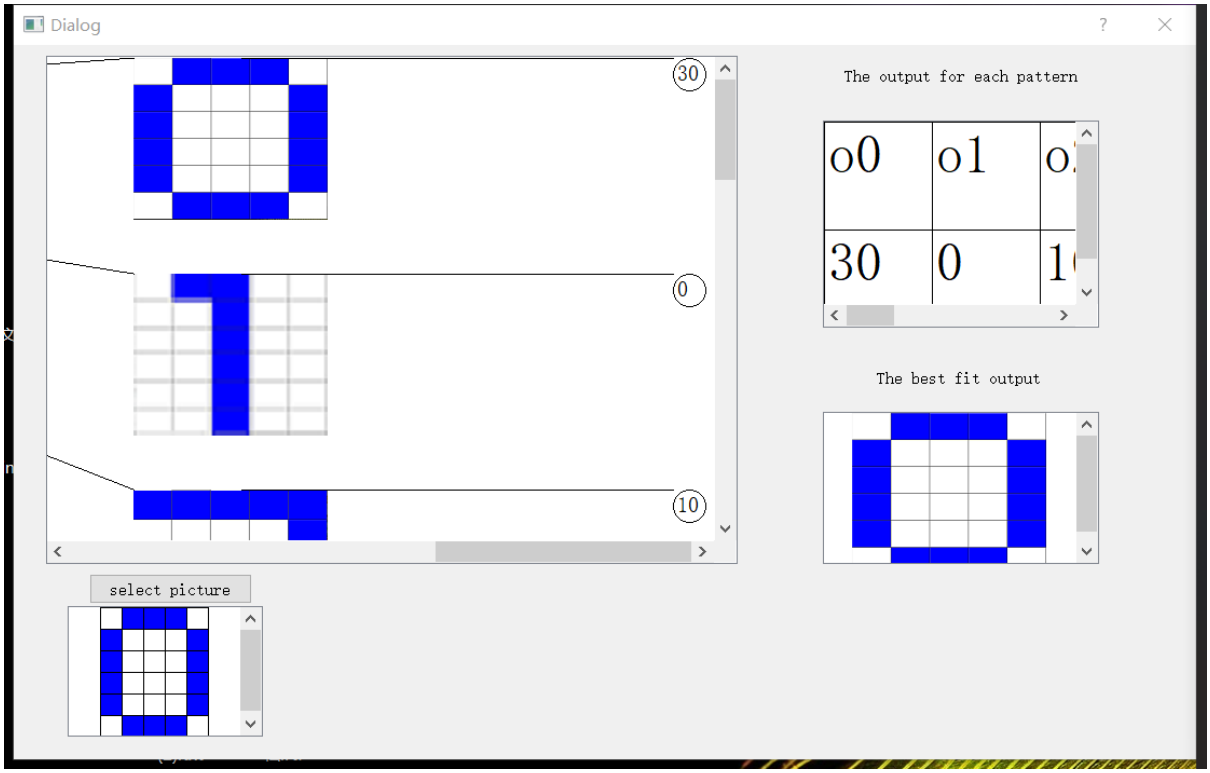

Figure 31: test of result

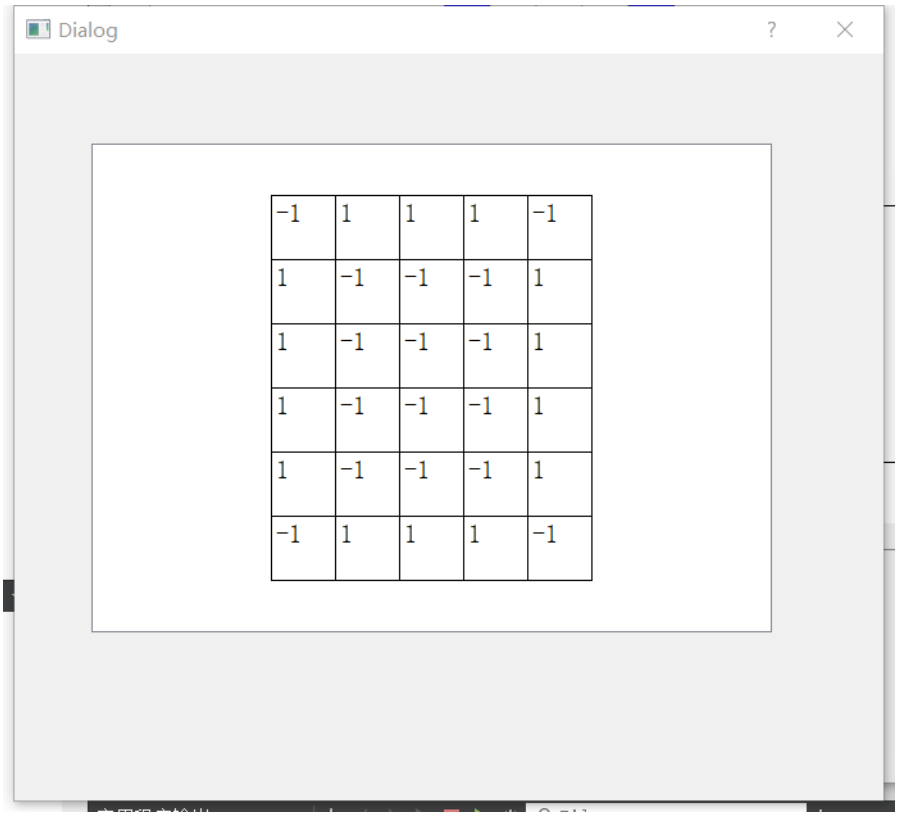

Figure 32: internal data of template 0

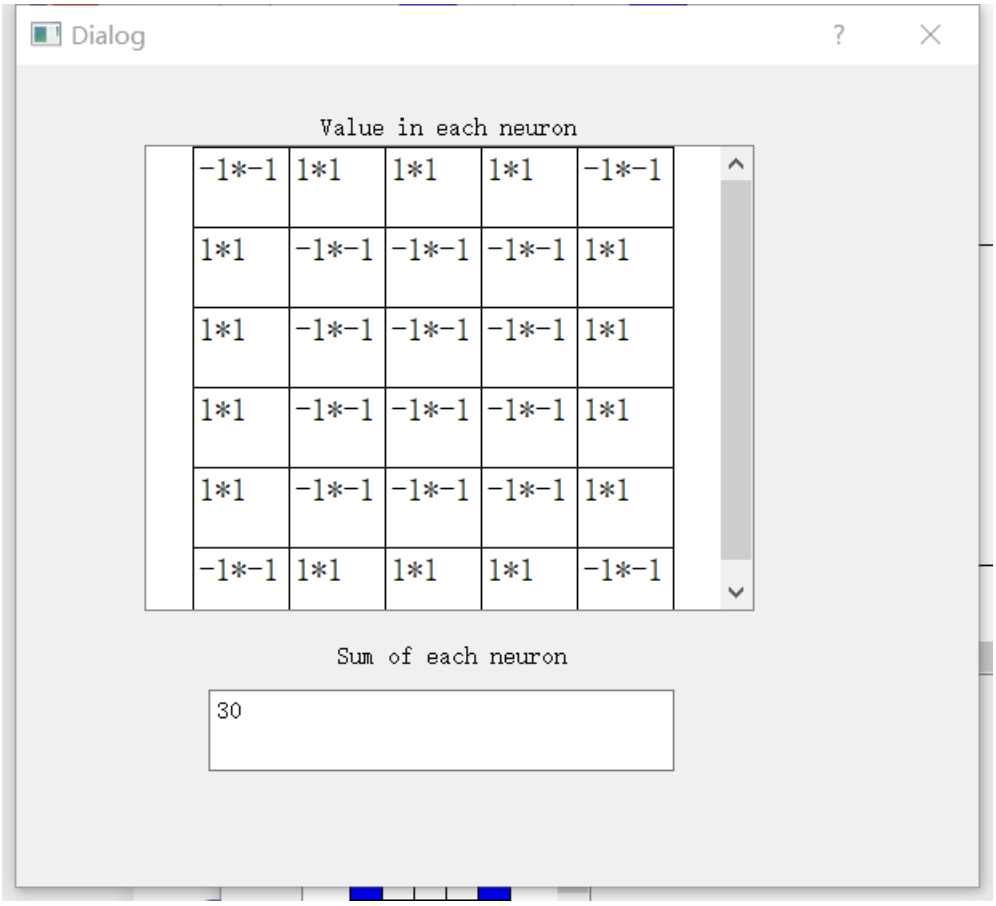

Figure 33: internal calculation of output 0

Table 15: the output with each template

The template	0	1	2	3	4	5	6	7	8	9
The output	30	0	10	10	-2	11	14	10	16	12

4.2 Evaluation

According to the response of the user, the strength and weakness are summarized, see Table.16.

Table 16: strength and weakness of the application

strength	weakness

Correct illustration of Hebb's rule	Unvivid interface
Has educational implication	The result of test and train is not related
Customized parameters	Doesn't give suggestion on parameters

5. Conclusion

Hebb's rule is a basic theory of artificial intelligence, since it was introduced by Donald Hebb, has been regarded as the neuron basis of unsupervised learning. However, as a preliminary theory, there was not many applications for students to learn and approach it. To deal with this problem, this project aims to design a program which explains the hebb's rule thoroughly by illustrating the process of applying hebb's rule in image recognition. The objects are illustrating the hebb's rule, verify the correctness of hebb's rule and explain the role of different parameters. The first object is achieved by constructing a application which use image recognition as an example, showing the update of each weight during the process. The second object is achieved by presenting the internal calculation of weights in a separate interface. The third object is achieved by allowing customized parameter and user can use different combinations of the parameter to recognize the difference.

Overall, the program is successful, it works as expected, also receive the positive review from the user. However, this program still has some drawbacks according to the user. First of all, the user interface needs to be beautified. In the second, the data could be visualized as bar or chart to make it clearer.

6. BCS Project Criteria & Self-reflection

In this chapter, the aspect of how to achieve the six outcomes expected by the chartered institute for IT is discussed. Those criteria reflect the capability of student to use the knowledge disseminated in the module and also give overview about successful and imperfect part of the project

6.1 Practical skill gained in the degree program

A lot of skills gained from class were included in this project. One example is the project management, as I referred in the section 4.1, in the planning stage, the waterfall model is adopted to run the project. The waterfall is taught in the model COMP201 software engineering, as one of the basic models running in the project management. The other example is the language used in the project. Qt language is the language I learned in the model ELEC362, it leaved me the impression of handy interface manipulation. That is why I choose the Qt as programming language.

6.2 Innovation and creativity

The innovation of this project is using the application of image recognition as an example to explain the process of hebb's rule to the user. And in the program, users are able to access to the internal calculation of the application which help them understand the theory thoroughly. Besides, users are allowed to customize their input data and set different parameters to get the different results from the program. It can boost the interest of students to the hebb's rule.

6.3 Synthesis of information, ideas and practices

The idea of using image recognition to illustrate hebb's rule is not only because image recognition is a popular area for the application of hebb's rule, but also because it is visualized clearly which impress users. Besides, Qt is an interface which is based language which provides a lot of stand-alone widgets to be used. Compared to other language, such as C or python, it saves a lot of time in editing the interface, which shorten my time of coding as a result.

6.4 The real need in wider context

At the present time, the unsupervised learning is becoming more and more popular in artificial intelligence. As the basis of unsupervised learning, Hebb's rule is critical for

students to learn. However, the conventional way to learn this knowledge is reading the book and attending the tutorial. Those methods are boring and lack of interaction. This project provides the student an interesting way to look through the all process of hebb's rule. Therefore, it can be valuable in the preliminary stage of unsupervised learning.

6.5 An ability to self-manage work

As for the management part. I made my schedule based on water fall model. Following the requirement-design-implement-verification-maintenance structure, I allocate 10 days for designing, 21 days for coding and 4 days for the review. By obeying the schedule, the project works smoothly.

6.6 Critical self-evaluation

The final year project is a good test for the final year student to evaluate and improve their skills and knowledges. For the good part, I fully use the knowledge of QT, project management and artificial intelligence which I learned from the class to make the project successfully. And I also plan the project in advance which help me have more time to prepare for the project. For the bad part, during the coding, I find that I haven't fully grasp the coding skill which cost me a lot of time to look up for the document. Secondly, I didn't look for the advice from the friend which make me stuck in one problem for a long time, then I realize that seeking for help from others is efficient in some condition.

7.Reference

[1] Y. Li, Y. Zhang and H. Wang, "Partial Parallel Interference Cancellation Multiuser Detection using Recurrent Neural Network Based on Hebb Learning Rule," 2006 6th World Congress on Intelligent Control and Automation, Dalian, China, 2006, pp. 2989-2992, doi: 10.1109/WCICA.2006.1712914.

[2] Y. Pan, "Development of Image Processing Software Based on Qt Creator," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control

Conference (ITNEC), Chengdu, China, 2019, pp. 2667-2671, doi: 10.1109/ITNEC.2019.8729186.

[3] P. Trivedi and A. Sharma, "A comparative study between iterative waterfall and incremental software development life cycle model for optimizing the resources using computer simulation," 2013 2nd International Conference on Information Management in the Knowledge Economy, Chandigarh, India, 2013, pp. 188-194.

[4] "*QGraphicsItem Class*". Accessed on: Apr. 4, 2021. [Online]. Available: https://doc.qt.io/qt-5/qgraphicsellipseitem.html

[5] O.Goffart , *How Qt Signals and Slots Work*, Woboq, Dec. 2012. Accessed on: Apr. 4, 2021. [Online]. Available: https://woboq.com/blog/how-qt-signals-slots-work.html

[6] S. Akio and Y. Masaki, "Design of a novel central pattern generator and the hebbian motion learning," *2009 IEEE Control Applications, (CCA) & Intelligent Control, (ISIC)*, 2009, pp. 1655-1660, doi: 10.1109/CCA.2009.5281012.

[7] I. Ahmad, P. P. Mondal and R. Kanhirodan, "Hebbian learning based FIR filter for image restoration," *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, 2005.*, 2005, pp. 726-730, doi: 10.1109/ISSPIT.2005.1577188.

8. Appendix: Software source code

```
original_Hebb::original_Hebb(int *a, int *weight, int size,int t,int C)
{
    int sum=0;
    int X=0;
//    qDebug()<<weight[2];
    for(int i=0;i<size;i++){
        sum=sum+a[i]*weight[i];

    }
//    qDebug()<<sum;
    if(sum>=t){
        X=1;
    }
    QVector<qreal> dealt_weight;
    for(int i=0;i<size;i++){
        dealt_weight.append(a[i]*C*X);
        weight[i]=weight[i]+dealt_weight[i];
    }

    this->X=X;

}
```

Figure 34: code of original Hebb's rule

```
class SelectPicture : public QDialog
{
    Q_OBJECT

public:
    explicit SelectPicture(QWidget *parent = nullptr);
    ~SelectPicture();
    void insertrec(QGraphicsScene *scene,int x, int y, int weight, int height);

signals:
    void senddata(int *data);

private slots:
    void on_buttonBox_accepted();

    void on_pushButton_clicked();

    void on_pushButton_2_clicked();
    int* return_data();
```

```
class Test : public QDialog
{
    Q_OBJECT

public:
    explicit Test(QWidget *parent = nullptr);
    ~Test();
    void addLineItem(QGraphicsScene *scene, int start_x,int start_y, int end_x, int end_y);
    int addValueItem(QGraphicsScene *scene, int *a, int *b, int x,int y);
    void addPixmap(QGraphicsScene *scene,MygraphicsItem *pictureItem, int x, int y,int *a,int *b);
    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    void getmaximage(int pos);
    void redraw();
private slots:
    void on_pushButton_clicked();
    void receive_data(int *data);
    void setdata(int *data);
```

```
void Test::on_pushButton_clicked()
{
    SelectPicture selec_dig(this);

    connect(&selec_dig,SIGNAL(senddata(int* )),this,SLOT(receive_data(int* )));
    selec_dig.exec();
}
```

Figure 35: the code of using signal and slot to enable the communication

```
Normalized_hebb::Normalized_hebb(int *a, double *weight, int size, int t, int C)
{
    QVector<double> old_weight,new_weight;
    for(int i=0;i<size;i++){
        old_weight.append(weight[i]);
    }
    double mod=0;
    for(int i=0;i<size;i++){
        mod+=weight[i]*weight[i];
        }
    mod=sqrt(mod);
    this->sqrtofsum=mod;
    for(int i=0;i<size;i++){
        weight[i]=weight[i]/mod;
    }
    double sum=0;
    int X=0;
    for(int i=0;i<size;i++){
        sum+=weight[i];
    }
    if(sum>t){
       X=1;
    }
    this->X=X;
     QVector<double> dealt_weight;
     for(int i=0;i<size;i++){
         dealt_weight.append(C*a[i]*X);
         weight[i]+=dealt_weight[i];
         QString mid=QString::number(weight[i],'f',3);
         weight[i]=mid.toDouble();
     }
    for(int i=0;i<size;i++){
        new_weight.append(weight[i]);
    }
    for(int i=0;i<size;i++){
        if(qAbs(new_weight[i]-old_weight[i])>max_delta){
            max_delta=qAbs(new_weight[i]-old_weight[i]);
        }
    }
    this->difference=max_delta;
    if(max_delta<theta){
        countinue=0;
```

Figure 36: code of normalized Hebb's rule

```
calculation::calculation(int *a, int *b,int sum):
    ui(new Ui::calculation)
{
    ui->setupUi(this);
    this->a=a;
    this->b=b;
    ui->graphicsView->setScene(scene);
    for(int x=0;x<5;x++){
        for(int y=0;y<6;y++){
             QRectF Rect(60*x,50*y,60,50);
             scene->addRect(Rect);

        }
    }
    for(int x=0;x<6;x++){
        for(int y=0;y<5;y++){
            QGraphicsTextItem *pItem1=new QGraphicsTextItem();
            QGraphicsTextItem *pItem2=new QGraphicsTextItem();
            QGraphicsTextItem *pItem3=new QGraphicsTextItem();
            QString number1=QString::number(a[x*5+y])+"*"+QString::number(b[x*5+y]);
            QString operator1="*";
            QString number2=QString::number(b[x*5+y]);
            pItem1->setPlainText(number1);
            QFont font = pItem1->font();
            font.setPixelSize(20);
            pItem1->setFont(font);
            pItem1->setPos(60*y,50*x);
            scene->addItem(pItem1);

        }
    }

    ui->textBrowser_2->setPlainText(QString::number(sum));
}
```

Figure 37: code of association

```
class myEllipseItem:public QGraphicsEllipseItem
{
public:

    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    void setData(int *a, int *b,int sum);
private:
    int *a;
    int *b;
    int sum;
};

#endif // MYELLIPSEITEM_H
```

```
class MygraphicsItem:public QGraphicsPixmapItem
{

public:
    void setData(int *data);
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event);
private:
    int* data;
};

#endif // MYGRAPHICSITEM_H
```

```
void MygraphicsItem::setData(int *data)
{
    this->data=data;
}

void MygraphicsItem::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    test_data dig(data);
    dig.exec();
}
```

Figure 38: the code of clickable template

```
void myEllipseItem::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    calculation dig(a,b,sum);
    dig.exec();

}

void myEllipseItem::setData(int *a, int *b, int sum)
{
    this->a=a;
    this->b=b;
    this->sum=sum;
}
```

Figure 39: the code of clickable association value

```
#include <QGraphicsRectItem>
#include <QBrush>
class MyRectItem:public QGraphicsRectItem
{
public:
    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    int flag=-1;
    int getflag();
    void setcolor(int flag);
};

#endif // MYRECTITEM_H
```

```
#include "myrectitem.h"



void MyRectItem::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    if(this->brush()!=QColor("blue")){
        this->setBrush(QColor("blue"));
        this->flag=1;
    }else{
        this->setBrush(QColor("white"));
        this->flag=-1;
    }
}

int MyRectItem::getflag()
{
    return flag;
}

void MyRectItem::setcolor(int flag)
{
    if(flag==1){
        this->setBrush(QColor("blue"));
    }else{
        this->setBrush(QColor("white"));
    }
}
```

Figure 40: the code of clickable element