บทที่ 5

การพัฒนาโปรแกรมและการทดสอบ

เครื่องมือที่ใช้ในการพัฒนา

1. เครื่องคอมพิวเตอร์ส่วนบุคคล
2. ระบบปฏิบัติการวินโดว์ เวอร์ชัน 3.1
3. Microsoft Visual C++ เวอร์ชัน 1.5
4. Microsoft Visual Basic เวอร์ชัน 4 แบบ 16 บิต

แฟ้มข้อมูลที่ใช้

เป็นแฟ้มข้อมูลแบบ Text File ทำให้ผู้ใช้สามารถใช้เครื่องมือซอฟต์แวร์อื่นที่จัดการกับ Text File ในการดูหรือแก้ไขได้

1. แฟ้มข้อมูลที่เก็บข้อมูลชุดการสอน

จะเก็บข้อมูลของแต่ละชุดข้อมูลการสอน เป็น 1 รายการ โดยที่จัดเก็บข้อมูลของแต่ละโหนดด้วยค่า (Floating Point) และแต่ละค่าข้อมูลจะเว้นด้วยช่องว่าง ตามตาราง 5.1 แสดงปัญหา XOR ทั้ง 4 รูปแบบ ซึ่งแฟ้มข้อมูลของชุดข้อมูลการสอน จะประกอบด้วย 4 รายการดังนี้

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

เมื่อ X_1 คือข้อมูลนำเข้าของโหนดที่ 1
X_2 คือข้อมูลนำเข้าของโหนดที่ 2
T คือค่าเป้าหมาย

X_1	X_2	T
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

ตาราง 5.1 แสดงชุดข้อมูลในการแก้ปัญหา XOR

2. แฟ้มข้อมูลที่เก็บค่าน้ำหนัก

จะเก็บชุดของค่าน้ำหนักทั้งหมดของทุกๆชั้น โดยที่เป็นชุดของค่าน้ำหนักของข้อมูลนำเข้า 1 โหนดไปยังแต่ละโหนดในชั้นถัดไป เช่น ข่ายงานการแก้ปัญหา XOR ที่มี 1 ชั้นแอบแฝง มีโครงสร้างแบบ 2-4-1 คือ ชั้นนำเข้าข้อมูลมี 2 โหนด ชั้นแอบแฝงมี 4 โหนด ชั้นผลลัพธ์มี 1 โหนด

ตามรูป 5.1 จะเก็บข้อมูลดังนี้

1 -0.639998 0.600000 -0.660000 -0.920000

1 -0.699998 -0.400000 -0.819998 0.159998

2 -1.725069

2 0.520711

2 -4.430139

2 -0.222553

<u>คำอธิบายความหมายข้อมูลในแฟ้มข้อมูลค่าน้ำหนัก</u>

บรรทัดแรก หมายถึงชุดของค่าน้ำหนักจากโหนดแรกในชั้นนำเข้าข้อมูลไปยังทุกๆโหนดในชั้นแอบแฝง

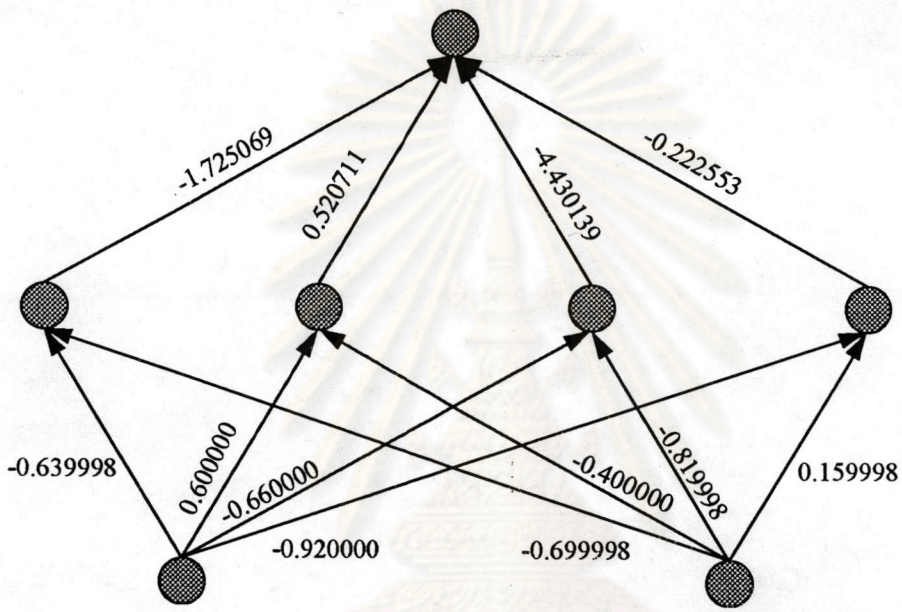
บรรทัดที่สอง หมายถึงชุดของค่าน้ำหนักจากโหนดที่สองในชั้นนำเข้าข้อมูลไปยังทุกๆโหนดในชั้นแอบแฝง

บรรทัดที่สาม หมายถึงชุดของค่าน้ำหนักจากโหนดแรกในชั้นแอบแฝงไปยังโหนดในชั้นผลลัพธ์

บรรทัดที่สี่ หมายถึงชุดของค่าน้ำหนักจากโหนดที่สองในชั้นแอบแฝงไปยังโหนดในชั้นผลลัพธ์

บรรทัดที่ห้า หมายถึงชุดของค่าน้ำหนักจากโหนดที่สามในชั้นแอบแฝงไปยังโหนดในชั้นผลลัพธ์

บรรทัดที่หก หมายถึงชุดของค่าน้ำหนักจากโหนดที่สี่ในชั้นแอบแฝงไปยังโหนดในชั้นผลลัพธ์



รูปที่ 5.1 แสดงโครงสร้างแบบ 2-4-1 พร้อมกับค่าน้ำหนักในแต่ละชั้น

3. แฟ้มข้อมูลที่เก็บค่าเฉลี่ยความผิดพลาดกำลังสอง

สำหรับค่าเฉลี่ยความผิดพลาดกำลังสอง ที่ได้จากการคำนวณในแต่ละรอบ จะทำการบันทึกลงแฟ้ม เพื่อให้ผู้ใช้สามารถนำไปใช้งานต่อ เช่นนำเสนอด้วยกราฟเพื่อจะได้เห็นถึงการเปลี่ยนแปลงค่าเฉลี่ยความผิดพลาดกำลังสองระหว่างการสอน ในแฟ้มข้อมูลนี้จะทำการบันทึก 2 ค่า คือค่าของรอบที่ทำการสอน และค่าเฉลี่ยความผิดพลาดกำลังสองในรอบนั้น เช่น

1,0.322397

2,0.276041

3,0.264396

4,0.259263

5,0.256227

6,0.252056

7,0.246637

8,0.239991

9,0.232136

10,0.222913

4. ชื่อแฟ้มข้อมูลที่เก็บชุดข้อมูลทดสอบ

จะเก็บข้อมูลคล้ายกับการเก็บในแฟ้มข้อมูลชุดการสอน แต่จะไม่มีการระบุค่าเป้าหมายในแฟ้มนี้ เช่นในการแก้ปัญหา XOR จะมีรูปแบบ ดังนี้

1 1

1 -1

-1 1

-1 -1

5. แฟ้มข้อมูลที่เก็บค่าผลลัพธ์

จะเป็นผลลัพธ์ของแต่ละโหนดในชั้นผลลัพธ์ที่ได้จากการคำนวณสำหรับชุดข้อมูลที่ใช้ทดสอบแต่ละชุด และจะนำผลลัพธ์นี้ไปวิเคราะห์อีกครั้งว่าได้ผลตามที่ตั้งเป้าหมายสำหรับชุดข้อมูลที่ใช้ทดสอบนี้หรือไม่ เช่น เมื่อกำหนดให้ชั้นผลลัพธ์มี 4 โหนด ข้อมูลในแฟ้มนี้จะมี 4 ค่า และขั้นแต่ละค่าด้วยเครื่องหมายจุลภาค เช่น

0.218660,0.644343,0.320555,0.575566

<u>ฟังก์ชันต่างๆในโปรแกรมการจำลองการทำงานของแบ็กพรอเพเกชัน</u>

ขั้นตอนการทำงานของโปรแกรมนี้ จะประกอบด้วย 2 ส่วนสำคัญคือ

1. ฟังก์ชันในการกำหนดค่าต่างๆที่ใช้ในระบบ
2. ฟังก์ชันที่ใช้ในขั้นตอนการสอนและทดสอบ

1. ฟังก์ชันในการกำหนดค่าต่างๆที่ใช้ในระบบ

ประกอบด้วยฟังก์ชันต่างๆดังนี้

1.1. ฟังก์ชันกำหนดโครงสร้างของระบบ

SetNetWork (int nMaxVector, int nLayer, int FAR *ai)

1.1.1 nMaxVector ระบุจำนวนชุดข้อมูลทั้งหมดที่ระบบจะสามารถจะเรียนรู้ได้ ที่จะนำไปจองค่าตัวแปรสำหรับเก็บข้อมูลที่จะใช้สอนทั้งหมด

1.1.2 nLayer กำหนดจำนวนชั้นของโครงสร้างที่ใช้แก้ปัญหา

1.1.3 *ai กำหนดจำนวนโหนดในแต่ละชั้น เป็นอาเรย์ของชั้นสามารถกำหนดได้สูงสุด 5 ชั้น เพื่อที่จะใช้จองค่าตัวแปรค่าน้ำหนักในแต่ละชั้น

```
Void  SetNetWork (int nMaxVector, int nLayer, int FAR *ai)  {
    netlayer.maxVector = nMaxVector;
    netlayer.numLayer = nLayer;
    for (int i=0 ; i<nLayer ; i++) {
        index[0] = short(i);
        netlayer.aiLayer[i] = *LPINT(VBArrayElement(ai,1,index));  } }
```

โดยที่ netlayer มีโครงสร้างแบบ NETLAYER ซึ่งมีโครงสร้างที่ประกอบด้วย 3 ตัวแปรและมีตัวชี้ชื่อ LPNETLAYER ชี้ไปยัง NETLAYER

```
typedef struct tagNETLAYER {
        int     maxVector;
        int     numLayer;
        int     aiLayer[6];
} NETLAYER;
typedef  NETLAYER  FAR *  LPNETLAYER;
```

1.2. ฟังก์ชันกำหนดพารามิเตอร์ที่ใช้ในระบบ

SetParam (float error_torerance, float learning_parameter, float alpha, float nf, long maxCycle, int nUpdate)

เป็นการส่งค่าพารามิเตอร์ให้กับระบบเพื่อนำไปใช้ในการขั้นตอนการสอนเท่านั้น

1.2.1 error_torerance กำหนดเงื่อนไขในการหยุดสอนเมื่อค่าเฉลี่ยความผิดพลาดกำลังสองจากการคำนวนน้อยกว่าค่านี้

1.2.2 learning_parameter กำหนดค่าอัตราการเรียนรู้

1.2.3 alpha กำหนดค่าโมเมนตัมที่ใช้ในการปรับปรุงน้ำหนัก

1.2.4 nf กำหนดค่ารบกวนที่มีผลต่อชุดข้อมูลนำเข้า

1.2.5 maxCycle กำหนดเงื่อนไขในการหยุดสอน เมื่อประมวลผลครบตามจำนวนรอบที่กำหนด

1.2.6 nUpdate กำหนดทุกๆจำนวนรอบให้แสดงค่าเฉลี่ยความผิดพลาดกำลังสอง ในลิสต์บอกซ์

```
void SetParam (float error_torerance, float learning_parameter, float alpha, float nf, long maxCycle, int nUpdate) {
        parameter.error_tolerance = et;
        parameter.beta = learning_parameter;
        parameter.alpha = alpha;
        parameter.NF = nf;
        parameter.maxCycle = maxCycle;
        parameter.updateInterval = nUpdate;
        }
```

1.3. ฟังก์ชันกำหนดชื่อแฟ้มข้อมูลต่างๆที่ใช้ในระบบ

SetFilename (LPCSTR szW, LPCSTR szO, LPCSTR szE, LPCSTR szTrain, LPCSTR szTest)

1.3.1 szW กำหนดชื่อแฟ้มข้อมูลที่เก็บชุดค่าน้ำหนัก เมื่อหยุดสอนจะบันทึกชุดค่าน้ำหนักสุดท้ายไว้เพื่อใช้ในขั้นตอนทดสอบต่อไป

1.3.2 szO กำหนดชื่อแฟ้มข้อมูลที่เก็บชุดค่าผลลัพธ์

1.3.3 szE กำหนดชื่อแฟ้มข้อมูลที่เก็บค่าเฉลี่ยความผิดพลาดกำลังสองในแต่ละรอบเพื่อให้สามารถนำเสนอค่านี้ในรูปแบบอื่นๆได้

1.3.4 szTrain กำหนดชื่อแฟ้มข้อมูลที่เก็บข้อมูลชุดการสอน

1.3.5 szTest กำหนดชื่อแฟ้มข้อมูลที่เก็บชุดข้อมูลทดสอบ

```
void SetFilename (LPCSTR szW, LPCSTR szO, LPCSTR szE, LPCSTR szTrain,
LPCSTR szTest) {
        lstrcpy (LPSTR(files.weights), szW);
        lstrcpy (LPSTR(files.output), szO);
        lstrcpy (LPSTR(files.error), szE);
        lstrcpy (LPSTR(files.training), szT);
        lstrcpy (LPSTR(files.test), szTest);  }
```

เป็นการ copy ค่า address ของ Long Pointer Charater String ให้กับค่า address ของ buffer ดังนั้นเมื่อต้องการรู้ค่าที่ files.weights ชี้ ต้องระบุเป็น &files.weights

1.4. ฟังก์ชันกำหนดชื่อของลิสต์บอกซ์ที่ต้องการให้แสดงค่าเฉลี่ยความผิดพลาดกำลังสอง

```
SetRecieveControl (HWND hwndLstDisp)
```

1.4.1 hwndLstDisp เป็นชื่อของออบเจกซ์ลิสต์บอกซ์ที่ต้องการ

เป็นการส่งชื่อของลิสต์บอกซ์ในโปรแกรมการประยุกต์ใช้งานมาให้กับระบบ เพื่อให้โปรแกรมรู้ว่าจะส่งค่าเฉลี่ยความผิดพลาดกำลังสองระหว่างการประมวลผลไปที่ลิสต์บอกซ์ใด โดยที่ฟังก์ชันนี้จะตรวจสอบว่าค่าที่ส่งมาเป็น handle window หรือไม่ และถ้าใช่จะตรวจสอบว่าเป็นลิสต์บอกซ์เท่านั้น โดยตรวจสอบที่ GetClassName มาเก็บไว้ที่ LPSTR(s) และถ้าส่งลิสต์บอกซ์มาจะเก็บชื่อไว้ที่ตัวแปรแบบโกลบอลชื่อ ::hwndLstDisp

```
void  SetRecieveControl (HWND hwndLstDisp) {
  if (IsWindow(hwndLstDisp)) {
     GetClassName (hwndLstDisp, LPSTR(s), 79);
              // เก็บ class name ของ hwndLstDisp  ไว้ที่ s
     if (!lstrcmp (LPCSTR(s), "ThunderListBox"))    // ตรวจสอบ s เป็นลิสต์บอกซ์
        ::hwndLstDisp = hwndLstDisp;
     else
        MessageBox(GetActiveWindow(),"ListBoxonly",NULL,
                MB_ICONEXCLAMATION);
  }
  else
     MessageBox (GetActiveWindow(), "hWnd not valid!", NULL,
                MB_ICONEXCLAMATION);
}
```

1.5. ฟังก์ชันส่งข้อความหยุดการสอน

```
SendInterruptMesg (void)
```

ฟังก์ชันนี้ไม่มีพารามิเตอร์

จะกำหนดให้ fInterrupt เป็นจริงเมื่อมีการกดให้หยุดการสอน และในช่วงการสอน เมื่อประมวลผลครบ 1 รอบจะมาตรวจสอบที่ค่านี้ว่าเป็นจริงหรือยัง เมื่อเป็นจริงแล้วจะทำการบันทึกค่าน้ำหนัก และจบโปรแกรม

```
extern "C"  void WINAPI EXPORT SendInterruptMesg (void)  {
     fInterrupt = TRUE;
}
```

2. ฟังก์ชันที่ใช้ในขั้นตอนการสอนและทดสอบ

float BackProp (int TrainMode, int StartWeights)

เป็นฟังก์ชันหลักของโปรแกรมจำลองการทำงานแบบแบ็กพรอเพเกชันที่ในขั้นตอนการสอนและขั้นตอนการทดสอบจะมาเรียกใช้ ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวดังนี้

กำหนดการทำงานเป็นขั้นตอนการสอนหรือขั้นตอนการทดสอบ

ถ้า TrainMode เป็น 1 คือขั้นตอนการสอน

ถ้า TrainMode เป็น 0 คือขั้นตอนการทดสอบ

กำหนดการที่มาของชุดค่าน้ำหนัก

ถ้า StartWeights เป็น 1 คือจะอ่านค่าน้ำหนักจากแฟ้มข้อมูลที่กำหนดใน szW

ถ้า StartWeights เป็น 0 คือจะสุ่มค่าน้ำหนักใหม่

ฟังก์ชันนี้สามารถอธิบายการทำงานได้ดังนี้

2.1 ขั้นตอนสร้างโครงสร้างตัวแปร

2.2 ขั้นตอนการเตรียมข้อมูล

2.3 ขั้นตอนคำนวณหาผลลัพธ์ (Forward Pass)

2.4 ขั้นตอนส่งกลับค่าความผิดพลาด (Backward Pass)

2.5 การตรวจสอบเมสเสจการหยุดสอน

2.6 การส่งค่าความผิดพลาดไปยังลิสต์บอกซ์

2.7 การคำนวณค่าเฉลี่ยความผิดพลาดกำลังสอง

2.1 ขั้นตอนสร้างโครงสร้างตัวแปร

เนื่องจากข่ายงานแบ็กพรอเพเกชัน จะมีลักษณะโครงสร้างเป็นชั้น และอาจมีชั้นแอบแฝงมากกว่า 1 ชั้นได้ และการทำงานในแต่ละชั้นมีลักษณะคล้ายกันทำให้สามารถกำหนดโครงสร้างแบบ Class ได้อย่างเหมาะสม

2.1.1. คลาสและตัวชี้

จะสร้างตัวชี้ (pointer) ชี้ไปยังชั้นของข้อมูลต่างๆดังนี้คือ ในชั้นนำเข้าข้อมูลจะเป็น layer_ptr(0) และชั้นแอบแฝงจะมีตัวชี้เป็น layer_ptr(1) กรณีที่มีมากกว่า 1 ชั้นแอบแฝงจะใช้ layer_ptr(2) , layer_ptr(3) และในชั้นผลลัพธ์จะใช้ตัวชี้ลำดับถัดไป ดังนั้นในกรณีมี 1 ชั้นแอบแฝงชั้นผลลัพธ์จะใช้ตัวชี้เป็น layer_ptr(2)

โดยที่ชั้นที่มีการทำงานแบบเดียวกันจะอยู่ในคลาส (Class) เดียวกัน คือให้ชั้นนำเข้าข้อมูล จะมีคลาสเป็น input_layer หรือ layer_ptr(0) ชั้นแอบแฝงจะมีคลาสเป็น output_layer และ middle_layer หรือ layer_ptr(1) ถึง layer_ptr(n-2) ส่วนชั้นผลลัพธ์จะมีคลาสเป็น output_layer หรือ layer_ptr(n-1) ในคลาสเดียวกันจะมีการกำหนดค่าข้อมูลและฟังก์ชันการคำนวณต่างๆไว้ ให้ชั้นที่อยู่ในคลาสเดียวกันสามารถเรียกใช้ได้

2.1.2. จำนวนโหนดในแต่ละชั้น

จำนวนโหนดในแต่ละชั้นจะจองตัวแปร 2 ตัวคือ num_inputs และ num_outputs เพื่อเก็บจำนวนโหนดดังนี้

ในชั้นนำเข้า num_inputs เป็น 0 และ num_outputs เป็นจำนวนโหนดในชั้นนำเข้าข้อมูล

ในชั้นแอบแฝง num_inputs เป็นจำนวนโหนดในชั้นนำเข้าข้อมูล และ num_outputs เป็นจำนวนโหนดในชั้นแอบแฝง

ส่วนในชั้นผลลัพธ์ num_inputs เป็นจำนวนโหนดในชั้นแอบแฝง และ num_outputs เป็นจำนวนโหนดในชั้นผลลัพธ์

2.1.3. การจองเนื้อที่สำหรับตัวแปรและการอ้างอิงในแต่ละชั้น

ในคลาส input_layer ชั้นนำเข้าข้อมูล จะจอง outputs และ orig_outputs เท่ากับจำนวนโหนดในชั้นนั้น

ในคลาส output_layer หรือชั้นผลลัพธ์และชั้นแอบแฝงจะจอง weights ,cum_deltas และ past_deltas เท่ากับผลคูณของจำนวนโหนดนำเข้า (num_inputs) และโหนดผลลัพธ์ (num_outputs) ของชั้นนั้น และจอง outputs, output_errors, expected_values เท่ากับจำนวนโหนดผลลัพธ์ของชั้นนั้น และจอง back_errors เท่ากับจำนวนโหนดนำเข้าของชั้นนั้น

```
weights        = new float[num_inputs*num_outputs];
output_errors  = new float[num_outputs];
```

```
back_errors     = new float[num_inputs];
outputs         = new float[num_outputs];
expected_values = new float[num_outputs];
cum_deltas      = new float[num_inputs*num_outputs];
past_deltas     = new float[num_inputs*num_outputs];
```

ในทุกๆชั้นแอบแฝงและชั้นผลลัพธ์จะอ้างอิงค่า outputs ในชั้นก่อนหน้า ไปยังค่า inputs ในชั้นถัดไป เพื่อให้เข้าใจง่ายและประหยัดตัวแปร ดังนี้

```
for (i=1; i<number_of_layers; i++)
    layer_ptr[i]->inputs = layer_ptr[i-1]->outputs;
```

เมื่อ number_of_layers เป็นจำนวนชั้นทั้งหมดที่ใช้ในระบบ ซึ่งอย่างน้อยคือ 3 ชั้น คือมี 1 ชั้นแอบแฝง

ความหมายของค่าความผิดพลาดในชั้นผลลัพธ์ หรือค่า output_errors คือ ความแตกต่างระหว่างผลลัพธ์ของแต่ละโหนดในชั้นนี้กับค่าเป้าหมาย และนำค่านี้ไปคำนวณหาค่าความผิดพลาดที่จะส่งกลับไปยังชั้นก่อนหน้า (ชั้นแอบแฝง) คือค่า back_errors ซึ่ง back_errors ในชั้นผลลัพธ์นี้จะถูกส่งเป็นค่าความผิดพลาดหรือ output_errors ในชั้นก่อนหน้า และในทำนองเดียวกันที่ชั้นแอบแฝงจะทำการคำนวณค่า back_errors เพื่อส่งไปยังชั้นก่อนหน้า ดังนั้นจึงอ้างอิงค่า outputs_errors ในชั้นก่อนหน้า ไปยังค่า back_errors ในชั้นถัดไป ดังนี้

```
for (i=1; i<number_of_layers-1 ; i++)
    ((output_layer *)layer_ptr[i])->output_errors =
((output_layer *)layer_ptr[i+1])->back_errors;
```

การจองเนื้อที่สำหรับข้อมูลทั้งหมดที่จะเข้าสู่การสอน เนื่องจากการสอน 1 รอบจะต้องสอนทุกๆชุดข้อมูล ซึ่งจะจองเนื้อที่เก็บค่าจำนวนจริง (Float) ซึ่งมีขนาด 4 ไบต์

```
i = layer_ptr[0]->num_outputs;                  // inputs
j = layer_ptr[number_of_layers-1]->num_outputs;  // outputs
```

```
long l = (long)nMaxVector;
if ((hgbl=GlobalAlloc(GPTR,(l*=(i+j)*sizeof(float)))) == NULL) {
    MessageBox (g_hwndActive, "insufficient memory for buffer", "Setup Network",
    MB_OK);
   ExpelApp ();
}
buffer = (float *)GlobalLock (hgbl);
```

ซึ่งจำนวนของโหนดในชั้นนำเข้าข้อมูลจะมีขนาดใหญ่สุดเท่ากับขนาดของ int หรือ 2 ไบต์ ซึ่งจะมีได้ถึง 32767 โหนด โดยที่แต่ละโหนดจะเก็บค่าจำนวนจริง 4 ไบต์ และจำนวนชุดข้อมูลทั้งหมดที่จะทำการสอนมีขนาดสูงสุด 32767 ชุดเช่นกัน โดยที่ขนาดข้อมูลรวมกันไม่เกิน 16 เมกะไบต์ เนื่องจากการใช้ GlobalAlloc( ) จะสามารถจองข้อมูลได้มากถึง 16 เมกะไบต์

2.2. ขั้นตอนการเตรียมข้อมูล

2.2.1. การกำหนดค่าน้ำหนัก

ในกรณีที่เป็นการสอนและกำหนดให้สุ่มค่าน้ำหนักใหม่ หรือ TrainMode เป็น 1 และ StartWeights เป็น 0 จะหาค่าน้ำหนักแบบสุ่มโดยจะได้ค่าเป็นเลขจำนวนจริงอยู่ระหว่าง -1 ถึง 1 โดยใช้

```
int num;
num=rand() % 100;
return  2*(float(num/100.00))-1;
```

ในกรณีที่เป็นการสอนและกำหนดให้อ่านค่าน้ำหนักจากแฟ้มข้อมูลเดิมที่เคยสอนไว้แล้ว หรือในการทดสอบ ก็จะใช้วิธีอ่านจากแฟ้มข้อมูล

2.2.2. การเตรียมข้อมูลนำเข้า

ในกรณีที่เป็นการสอน ชุดข้อมูลนำเข้าจะมีขนาดใหญ่กว่าในกรณีที่เป็นการทดสอบ เพราะจะมีกลุ่มข้อมูลที่เป็นค่าเป้าหมายด้วย

เนื่องจากการสอนของแบ็กพรอเพเกชัน จะต้องทำการประมวลผลกับทุกๆชุดข้อมูลการสอน จึงต้องนำข้อมูลจากแฟ้มข้อมูลที่เก็บชุดการสอนหรือทดสอบมาเก็บข้อมูลทั้งหมดไว้ในบัฟเฟอร์

```
ins  = layer_ptr[0]->num_outputs;
outs = layer_ptr[number_of_layers-1]->num_outputs;
if (training == TRAINMODE_ON)
     veclength = ins+outs;
else
     veclength = ins;
while ((count<nMaxVector)&&(!inputfile->Eof())) {
    l = (long)count * veclength;
    for (i=0 ; i<veclength ; i++) {
         if (inputfile->GetFloat(&buffer[l+i])) return -1;
    }
    count++;
}
```

ซึ่งจำนวนของโหนดในชั้นนำเข้าข้อมูลจะมีขนาดใหญ่สุดเท่ากับขนาดของ int หรือ 2 ไบต์ ซึ่งจะมีได้ถึง 32767 โหนด และจำนวนโหนดทั้งหมดที่จะนำมาสอนหรือผลรวมของโหนดจากทุกๆชุดการสอนมีจำนวนได้มากเท่ากับค่า long คือ 2,147,483,647 ซึ่งจะเป็นจำนวนชุดข้อมูลการสอนได้ไม่เกิน 32767 ชุดการสอน

2.3. ขั้นตอนคำนวณหาผลลัพธ์

จะทำการคำนวณหาผลลัพธ์ เพื่อส่งไปเป็นข้อมูลนำเข้าของชั้นถัดไป โดยจะทำการคำนวณผลลัพธ์ทีละชั้นตั้งแต่ชั้นนำเข้าข้อมูล

ในชั้นนำเข้าข้อมูล หรือ layer_ptr(0) จะปรับปรุงค่าข้อมูลนำเข้าจริงด้วยค่า noise_factor เพื่อปรับค่าข้อมูลนำเข้าในกรณีที่ข้อมูลนำเข้าอาจมีการถูกรบกวน ก่อนถูกส่งเข้ามาทำการประมวล

ผลเช่น จดจำใบหน้าคนที่บางครั้งยิ้มบางครั้งโกรธ การเพิ่มค่า noise_factor นี้เพื่อปรับค่าข้อมูลนำเข้าใหม่ โดยใช้

```
for (i=0; i<num_outputs; i++)
    outputs[i] = orig_outputs[i] * (1+ noise_factor * randomweight(0));
```

เมื่อ num_outputs เป็นจำนวนโหนดหรือโหนดในชั้นนำเข้าข้อมูล และrandomweight(0) เป็นการสุ่มค่าน้ำหนัก ซึ่งในกรณีนี้จะได้ค่าสุ่มเป็นเลขจำนวนจริงอยู่ระหว่าง -1 ถึง 1 โดยใช้

```
int num;
num=rand() % 100;
return  2*(float(num/100.00))-1;
```

ในกรณีที่ข้อมูลนำเข้าเป็นข้อมูลที่ไม่มีการถูกรบกวนเช่น การสอนให้แก้ปัญหา XOR ซึ่งต้องใส่ข้อมูลนำเข้าที่แน่นอน จึงไม่จำเป็นต้องปรับปรุงค่าข้อมูลนำเข้า ในโปรแกรมนี้เราสามารถกำหนดค่า noise_factor ได้ ถ้ากำหนดให้มีค่าเป็น 0 นั่นคือ จะไม่มีการปรับปรุงค่าข้อมูลนำเข้า

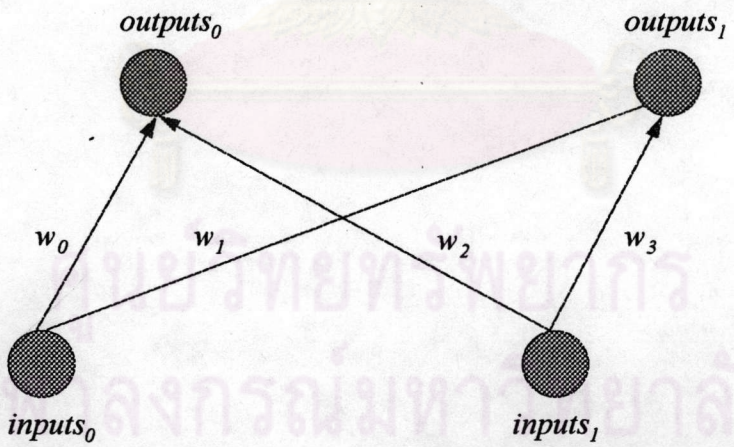
ในชั้นแอบแฝงและชั้นผลลัพธ์ จะทำคำนวณหาค่าผลลัพท์เพื่อส่งไปเป็นข้อมูลนำเข้าในชั้นถัดไป การคำนวณจะมีลักษณะตามรูป 5.2 คือการหาค่าผลลัพท์เพื่อนำมาเป็นค่าข้อมูลนำเข้าในชั้นแอบแฝงของโหนดหนึ่งๆจะมาจากทุกๆโหนดในชั้นนำเข้าข้อมูลกับค่าน้ำหนักของแต่ละโหนดในชั้นนำเข้าข้อมูลที่ไปยังโหนดหนึ่งในชั้นแอบแฝง และจะใช้วิธีเดียวกันในการหาค่าข้อมูลนำเข้าของทุกโหนดในชั้นแอบแฝง ในชั้นแอบแฝงถัดไปก็จะมีการทำงานแบบเดียวกัน และเมื่อเป็นชั้นแอบแฝงสุดท้ายจะส่งผลลัพท์ไปเป็นข้อมูลนำเข้าในชั้นผลลัพธ์ โดยในแต่ละชั้นจะทำการคำนวณดังนี้

```
int i,j,k;
float accumulator=0.0;
for (j=0; j<num_outputs; j++)
{
    for (i=0; i<num_inputs; i++)
    {
```

```
            k=i*num_outputs;
            outputs[j]=weights[k+j]*(*(inputs+i));
            accumulator+=outputs[j];
        }
    // use the sigmoid squash function
    outputs[j]=squash(accumulator);
    accumulator=0;
    }
```

โดยที่ squash(accumulator) หรือฟังก์ชันซิกมอยด์ จะส่งค่า accumulator ให้กับ input และทำการคำนวณ

```
return (float)(1/(1+exp(-(double)input)));
```



$$outputs_j = F(SUM(w_{ij} * inputs_i))$$

รูปที่ 5.2 แสดงตัวอย่างการหาผลลัพธ์ในแต่ละโหนด $outputs_j$

2.4 ขั้นตอนส่งกลับค่าความผิดพลาด

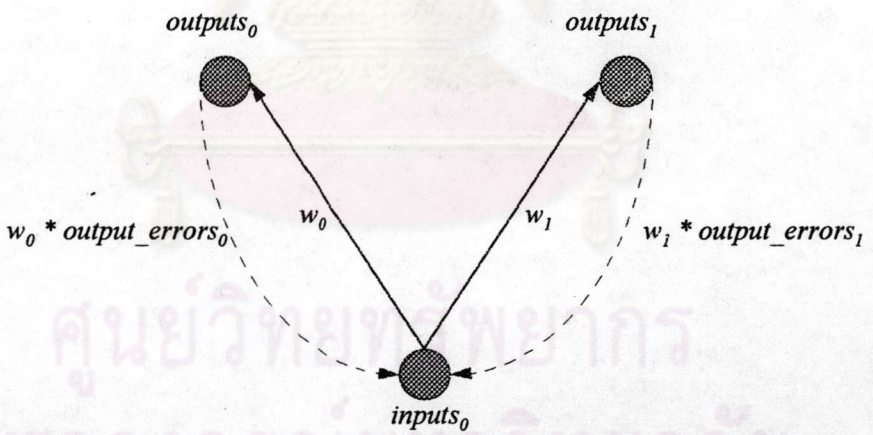
มีจุดประสงค์เพื่อปรับปรุงชุดค่าน้ำหนัก ถ้าค่าเฉลี่ยความผิดพลาดกำลังสองในรอบนั้นยังมากอยู่ ซึ่งประกอบด้วย การหาค่าความผิดพลาดเพื่อนำมาใช้ปรับปรุงค่าน้ำหนัก และการปรับปรุงค่าน้ำหนัก

2.4.1 การคำนวณหาค่าความผิดพลาด

จะทำการคำนวณหาค่าความผิดพลาด (output_errors) ในชั้นผลลัพธ์ ซึ่งคือความต่างระหว่างค่าเป้าหมายกับค่าผลลัพธ์ และส่งค่าผลรวมของค่าความต่างระหว่างค่าเป้าหมายกับค่าผลลัพธ์นี้ (error) ซึ่งเป็นค่าความผิดพลาดของ 1 ชุดข้อมูลเท่านั้น กลับไปเพื่อหาค่าความผิดพลาดเฉลี่ยกำลังสอง และคำนวณหาค่าความผิดพลาด (back_errors) ที่จะส่งกลับไปยังชั้นก่อนหน้าคือชั้นแอบแฝง โดยที่ back_errors คือค่าความผิดพลาดที่ถูกอิทธิพลจากค่าน้ำหนักต่างๆที่ส่งจากโหนดตัวหนึ่งในชั้นก่อนหน้าไปยังทุกๆโหนดในชั้นปัจจุบัน กับอิทธิพลจากค่าข้อมูลจากโหนดนั้น ตามรูป 5.3 โดยค่า Back_errors นี้ก็คือค่า output_errors ในชั้นก่อนหน้านั่นเอง ซึ่งจะมีการคำนวณดังนี้

```
int i, j, k;
float accumulator=0;
float total_error=0;
for (j=0; j<num_outputs; j++)
{
    output_errors[j] = expected_values[j]-outputs[j];
    total_error+=output_errors[j];
}
error=total_error;              // ส่งกลับค่า error นี้เพื่อไปคำนวณค่าเฉลี่ยกำลังสอง

//  ในทุกชั้นจะทำขั้นตอนต่อไปนี้ยกเว้น ชั้นนำเข้าข้อมูล
for (i=0; i<num_inputs; i++)
```

```
{
        k=i*num_outputs;
        for (j=0; j<num_outputs; j++)
        {
            back_errors[i]=weights[k+j]*output_errors[j];
            accumulator+=back_errors[i];
        }
        back_errors[i]=accumulator;
        accumulator=0;
    back_errors[i]*=(*(inputs+i))*(1-(*(inputs+i)));      // และเป็น output_errors ในชั้นก่อน
หน้า
    }
}
```



รูปที่ 5.3 แสดงตัวอย่างการหาค่าความผิดพลาดที่จะส่งกลับให้แต่ละโหนด

2.4.2 การปรับปรุงค่าน้ำหนัก

เมื่อค่าเฉลี่ยความผิดพลาดกำลังสองยังมากกว่าค่าที่ยอมรับได้ และจำนวนรอบในการสอนยังไม่ถึงที่กำหนดไว้ จะทำการปรับปรุงค่าน้ำหนักใหม่แต่ละชั้นสำหรับชุดข้อมูลถัดไป โดยที่จะปรับปรุงค่าน้ำหนักตามค่าอัตราการเรียนรู้ และค่าโมเมนตัมของการปรับปรุงค่าน้ำหนักในรอบที่ผ่านมา ดังนี้

```
int i, j, k; float delta;
// learning law: weight_change = beta*output_error*input + alpha*past_delta
for (i=0; i<num_inputs; i++) {
    k = i*num_outputs;
    for (j=0 ; j<num_outputs ; j++) {
        delta = beta * output_errors[j] * (*(inputs+i)) + alpha * past_deltas[k+j];
        weights[k+j] += delta;
        cum_deltas[k+j] += delta; // current cycle
    } }
```

เมื่อกำหนดโมเมนตัมเป็น 0 แสดงว่าการปรับปรุงค่าน้ำหนักในรอบที่ผ่านมาไม่มีผลกระทบต่อการปรับปรุงค่าน้ำหนักในครั้งนี้

2.5 การตรวจสอบเมสเสจการหยุดสอน

```
if (backp.get_training_value() == TRAINMODE_ON) {
    while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
```

เมื่อมีการส่งเมสเสจให้หยุดสอนจะมีการเก็บเมสเสจนั้นไว้ใน queue และระบบจะดึงเมสเสจนั้นมาใช้เมื่อทำการประมวลผลครบ 1 รอบแล้ว และเมื่อตรวจสอบว่ามีเมสเสจจะไม่จบโปรแกรมทันที แต่จะทำการบันทึกค่าน้ำหนักไว้เพื่อใช้ในขั้นตอนทดสอบ

2.6 การส่งค่าความผิดพลาดไปยังลิสต์บอกซ์

จะตรวจสอบเมื่อถึงรอบที่กำหนด เช่น ทุกๆ 5 รอบ จะทำการแสดงค่าของรอบและค่าเฉลี่ยความผิดพลาดกำลังสอง ดังนี้

```
if (++interval == parameter.updateInterval) {
    // ตรวจสอบจำนวนรอบที่จะแสดงค่าในลิสต์บอกซ์
  char s[80], buffer[50], sCycle[100]; int dec, sign, h, k; RECT rc;
  wsprintf (sCycle, "%d\t\t", total_cycles);
    // เก็บจำนวนรอบและต่อด้วยระยะ tab 2 ครั้งไว้ที่ sCycle
  lstrcpy (LPSTR(s), (LPCSTR)_fcvt(double(avgerr_per_pattern), 25, &dec, &sign));
  ftoa (buffer, s, dec, sign);
    // แปลงค่าเฉลี่ยความผิดพลาดกำลังสองให้เป็นค่า text เก็บไว้ที่ buffer
  lstrcat (LPSTR(sCycle), LPCSTR(buffer));          // ต่อค่า buffer เข้ากับ sCycle
  SendMessage (hwndLstDisp, LB_ADDSTRING, 0, LPARAM(LPCSTR(sCycle)));
    // ส่งค่า sCycle ไปที่ลิสต์บอกซ์
  GetClientRect (hwndLstDisp, &rc);
    // หาค่าขอบเขตของลิสต์บอกซ์ เช่น rc.top และ rc.bottom
  h = (int)SendMessage (hwndLstDisp, LB_GETITEMHEIGHT, 0, 0L);
    // หาความสูงของข้อมูลในลิสต์บอกซ์
  if ((k=(int)SendMessage(hwndLstDisp,LB_GETCOUNT,0,0L)) != LB_ERR)
    // นับจำนวนข้อมูลในลิสต์บอกซ์
   if ((rc.bottom-rc.top)/h < k) SendMessage (hwndLstDisp, LB_SETTOPINDEX, k-(rc.
  bottom-rc.top)/h, 0L);
    // ส่งค่าข้อมูลแรกที่แสดงในลิสต์บอกซ์ หรือทำการ scroll ลิสต์บอกซ์
  interval = 0; }
```

2.7. การคำนวณค่าเฉลี่ยความผิดพลาดกำลังสอง

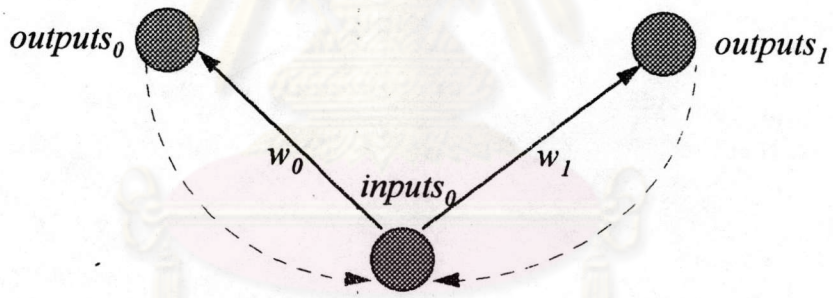
ตามรูป 5.4 จะหาค่าเฉลี่ยความผิดพลาดกำลังสอง และทำการเปรียบเทียบกับค่าที่ตั้งไว้สำหรับตัดสินว่าจะหยุดการสอนหรือไม่ จากแต่ละ Pattern ที่เข้ามาสู่ขั้นตอนการสอนแล้วได้ค่าผลลัพธ์ที่ต่างจากค่าเป้าหมายที่ต้องการ นั่นคือ

$$\text{ค่าความผิดพลาดกำลังสอง} = [\ \Sigma(\ \text{ค่าเป้าหมาย} - \text{ค่าผลลัพธ์จากการคำนวณ}\ )\ ]^2$$

เมื่อแต่ละ Pattern ผ่านไปจะคำนวณค่าความความผิดพลาดกำลังสองไว้ พร้อมกับนับจำนวน Pattern ไว้ เพื่อหาค่าความผิดพลาดเฉลี่ยจาก

$$\sqrt{\text{ผลรวมค่าความผิดพลาดกำลังสอง}\ /\ \text{จำนวน Pattern}}$$

$$output\_errors_j = expected\_value_j - outputs_j$$



$$total\_error_p = SUM\ (\ output\_errors_j\ )$$

$$MSE = SQRT\ (sum(total\_error_p)^2\ )\ /\ pattern\_per\_cycle$$

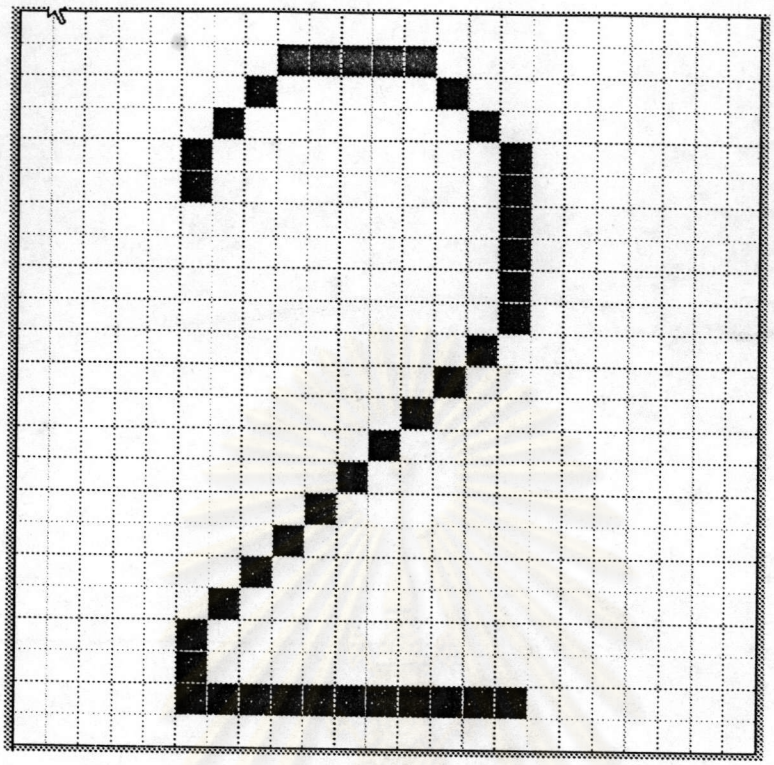
รูปที่ 5.4 แสดงตัวอย่างการหาค่าเฉลี่ยความผิดพลาดกำลังสอง

การพัฒนาโปรแกรมเพื่อทดสอบเครื่องมือซอฟต์แวร์

ในวิทยานิพนธ์นี้จะทดสอบโปรแกรมจำลองการทำงานของนิวรอลเนตเวิร์กที่ใช้ต้นแบบแบ็กพรอเพเกชัน โดยการประยุกต์สอนระบบให้เรียนรู้จดจำตัวเลข 0 - 9 ได้โดยจะสร้างกริดขนาด 23 x 23 ซึ่งจะได้จำนวนโหนดในชั้นนำเข้าข้อมูลขนาด 529 โหนด และในชั้นแอบแฝงมีขนาด 10 โหนด เพื่อสามารถแจกแจงรูปแบบทั้ง 10 ได้ และในชั้นผลลัพธ์จะมีขนาด 4 โหนดเพื่อให้สามารถแทนค่าข้อมูลผลลัพธ์ได้ทั้ง 10 รูปแบบดังตาราง 5.2

ค่าเป้าหมาย				ความหมาย
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	0

ตารางที่ 5.2 แสดงชุดของค่าเป้าหมายและความหมาย



รูปที่ 5.5 แสดงกริดของข้อมูลตัวเลขสอง

<u>การเตรียมชุดข้อมูลนำเข้า</u>

จะแปลงค่าข้อมูลจากกริดเป็นชุดข้อมูลนำเข้า ในที่นี้ใช้แกน X เป็นหลักและที่เซลล์ใดที่มีการคลิกจะให้เปลี่ยนสีและแทนค่าด้วย 1 ส่วนเซลล์ใดที่ไม่มีการคลิกให้แทนค่าด้วย 0 ดังนั้นตามรูป 5.5 แสดงกริดของข้อมูลเลขสอง จะแทนค่าเป็นข้อมูลตัวเลขสำหรับชุดข้อมูลนำเข้าได้ตามรูปที่ 5.6 ซึ่งชุดข้อมูลกลุ่มนี้เป็นตัวอย่างข้อมูลนำเข้ารูปแบบหนึ่งของค่าเลข 2 และจะมีชุดข้อมูลค่าเป้าหมายเป็น 0 0 1 0 โดยจะเก็บชุดข้อมูลตัวอย่างข้อมูลนำเข้า และชุดข้อมูลค่าเป้าหมาย นี้ลงแฟ้มข้อมูล เมื่อต้องการสอนให้เรียนรู้ตัวเลขทั้ง 10 รูปแบบ คือ 0 - 9 จะต้องมีชุดข้อมูลที่ส่ง ไปสอนอย่างน้อย 10 รูปแบบ เพื่อให้ระบบเรียนรู้ได้

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 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 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 0 0 0 0

รูปที่ 5.6 แสดงชุดข้อมูลนำเข้าของเลขสอง

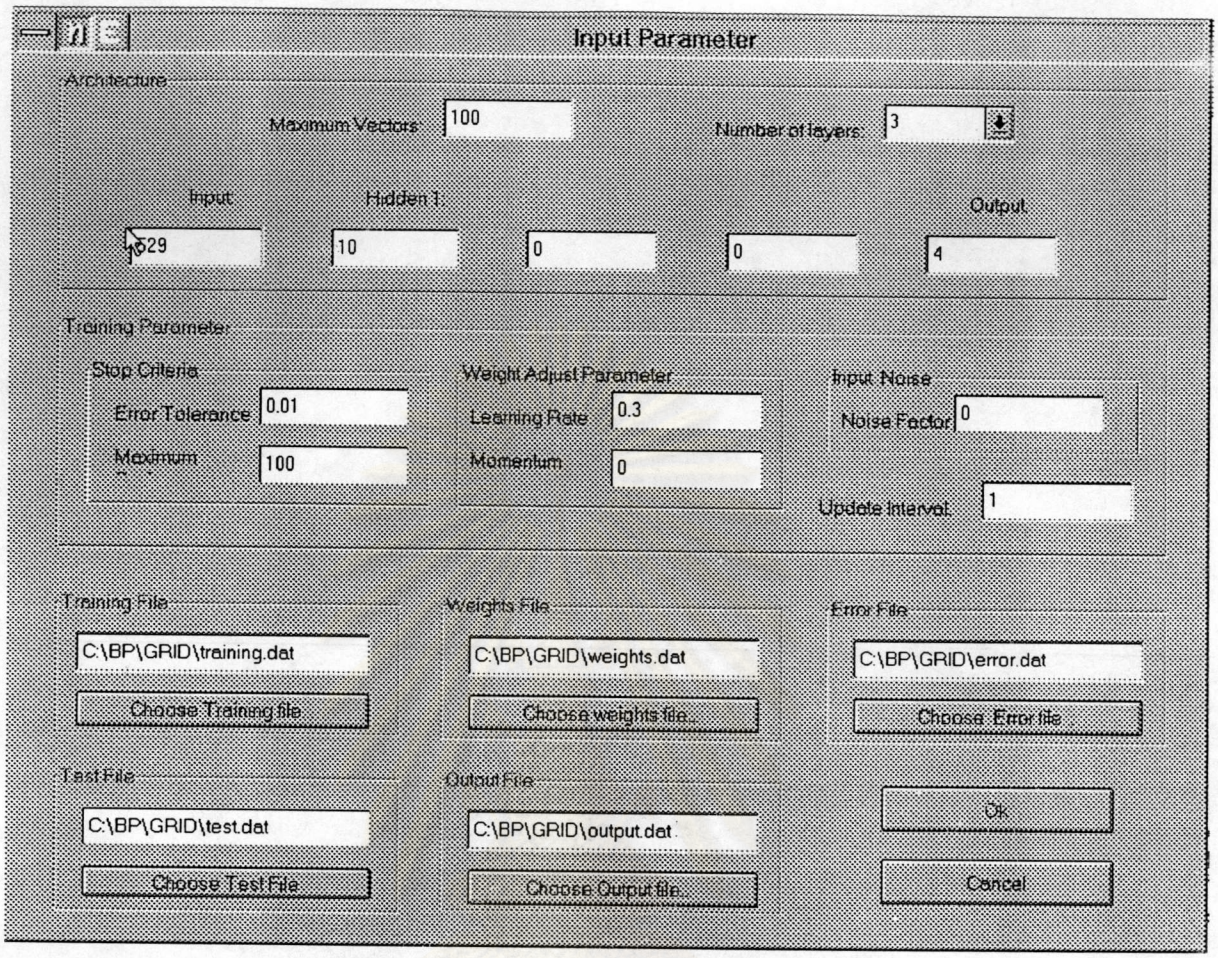
การอ้างฟังก์ชันต่างๆในวิชวลเบสิค

สามารถอ้างฟังก์ชันเหล่านี้ได้ในส่วนของฟอร์ม (Form) หรือ โมดูล (Module) ดังนี้

```
Declare Sub SetParam Lib "bp.dll" (ByVal et As Single, ByVal beta!, ByVal alpha!, ByVal NF!, ByVal MaxCycle&, ByVal nUpdate%)

Declare Sub SetNetWork Lib "bp.dll" (ByVal nMaxVector%, ByVal nLayer%, ByRef ai() As Integer)

Declare Sub SetFilename Lib "bp.dll" (ByVal szW$, ByVal szO$, ByVal szE$, ByVal szT$, ByVal szTest$)

Declare Sub SetRecieveControl Lib "bp.dll" (ByVal hwndDispListBox%)

Declare Sub SendInterruptMesg Lib "bp.dll" ()

Declare Function BackProp! Lib "bp.dll" (ByVal iTrainMode%, ByVal StartWeight%)
```

การกำหนดค่าพารามิเตอร์และโครงสร้าง

การกำหนดในส่วนนี้มีความจำเป็นอย่างยิ่ง ซึ่งจะต้องมีการกำหนดให้สอดคล้องกับการแก้ปัญหาหนึ่งๆ จะแบ่งออกเป็น 3 กลุ่ม คือ การกำหนดโครงสร้าง การกำหนดพารามิเตอร์ในการสอน และการกำหนดชื่อไฟล์ต่างๆที่ใช้ได้เอง ซึ่งจะมีหน้าตาดังรูป 5.7

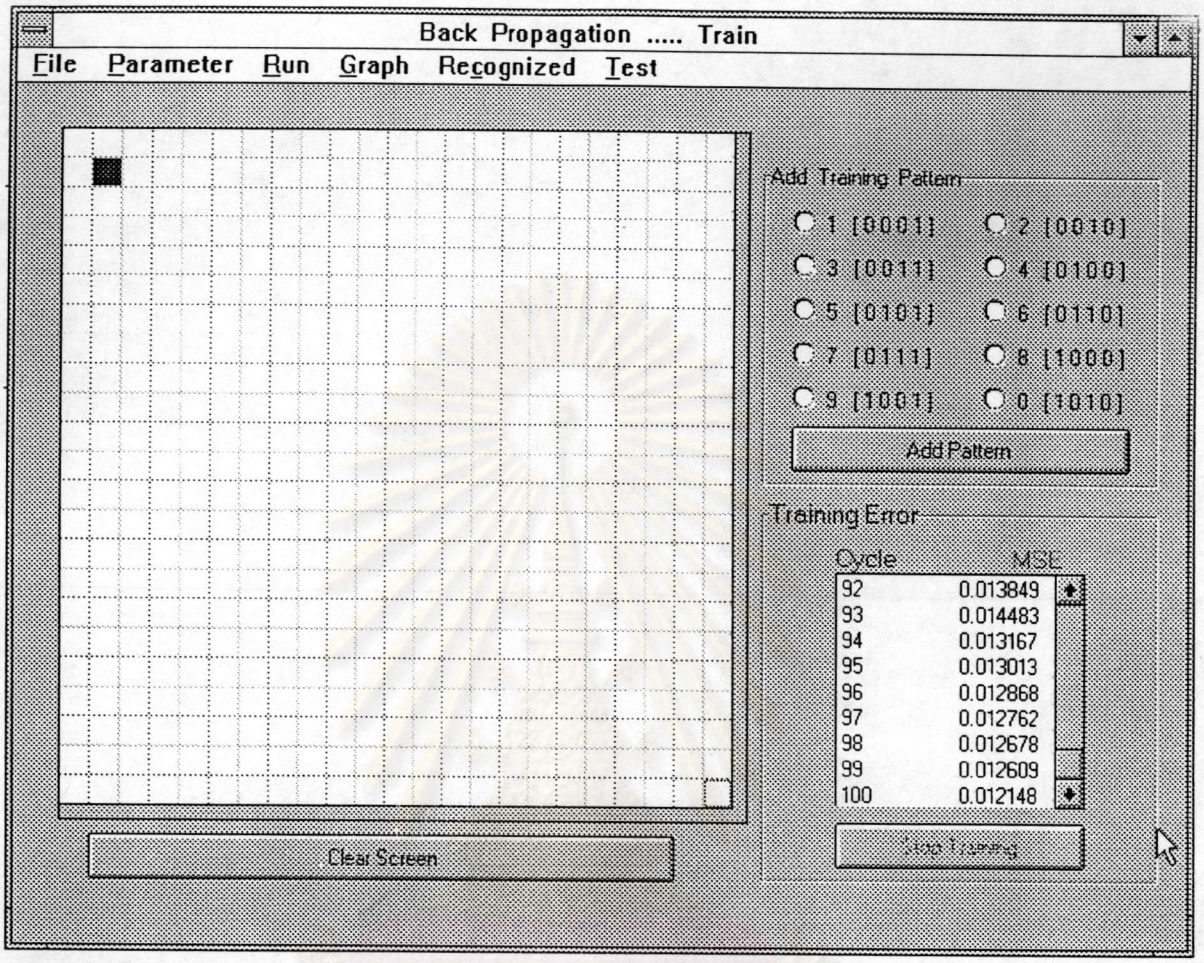


รูปที่ 5.7 แสดงจอภาพในการกำหนดพารามิเตอร์

<u>การสอน</u>

จะทำการเรียกใช้ฟังก์ชัน BackProp (1,0) เมื่อต้องการสุ่มค่าน้ำหนักใหม่ และเรียกใช้ฟังก์ชัน BackProp (1,1) เมื่อต้องการให้ดึงค่าน้ำหนักเก่าจากแฟ้มข้อมูลที่กำหนดในการรับค่าพารามิเตอร์ตามรูป 5.7 และแฟ้มข้อมูลที่ใช้ในการสอนจะอ่านจากค่าที่กำหนดไว้ เช่นกัน

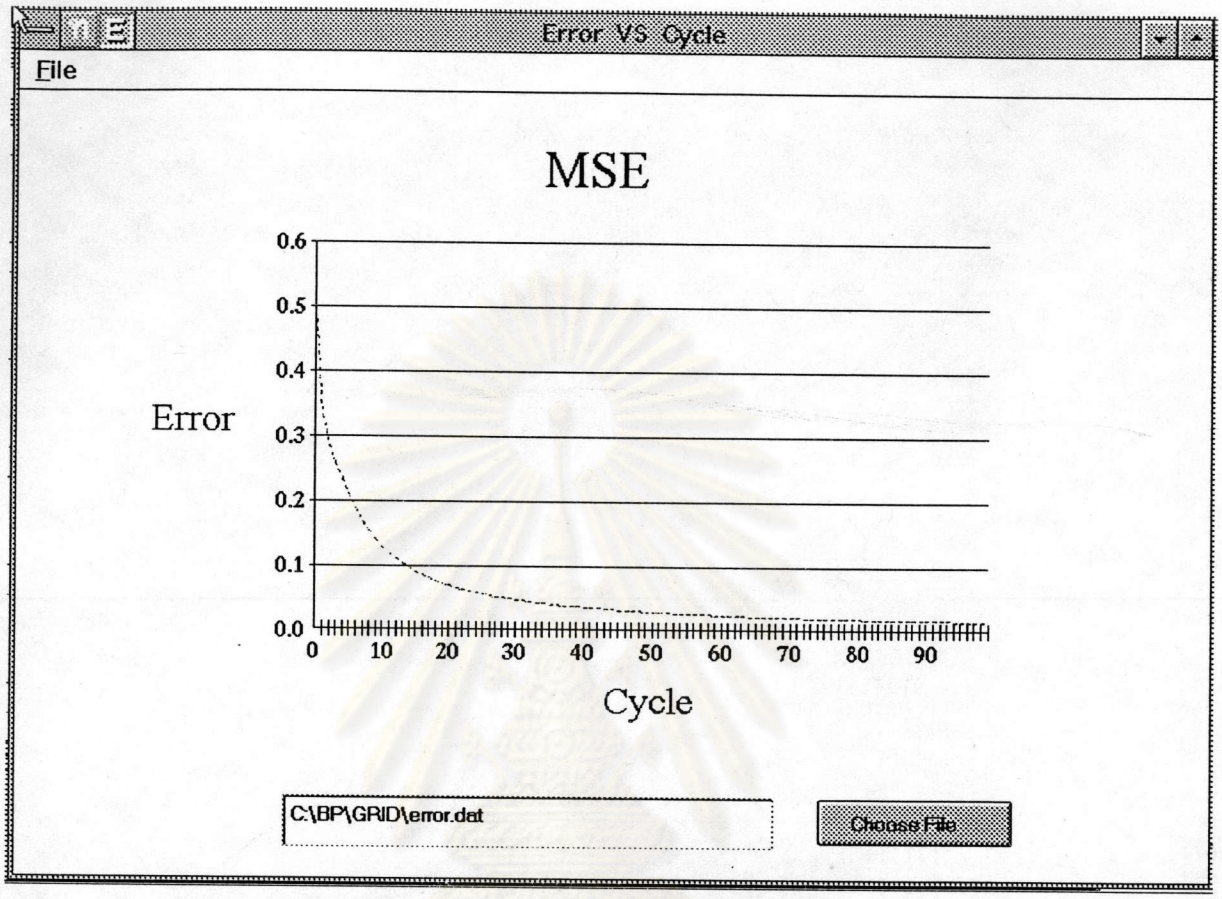
ในรูป 5.8 แสดงจอภาพในการสอน ในส่วนนี้จะสามารถเพิ่มชุดข้อมูลที่จะใช้สอนได้ และจะแสดงผลค่าเฉลี่ยความผิดพลาดกำลังสองในแต่ละรอบในลิสต์บอกซ์ และเปิดโอกาสให้ผู้ใช้สามารถหยุดการสอนได้



รูปที่ 5.8 แสดงจอภาพในขณะสอน

การสร้างกราฟของความผิดพลาด

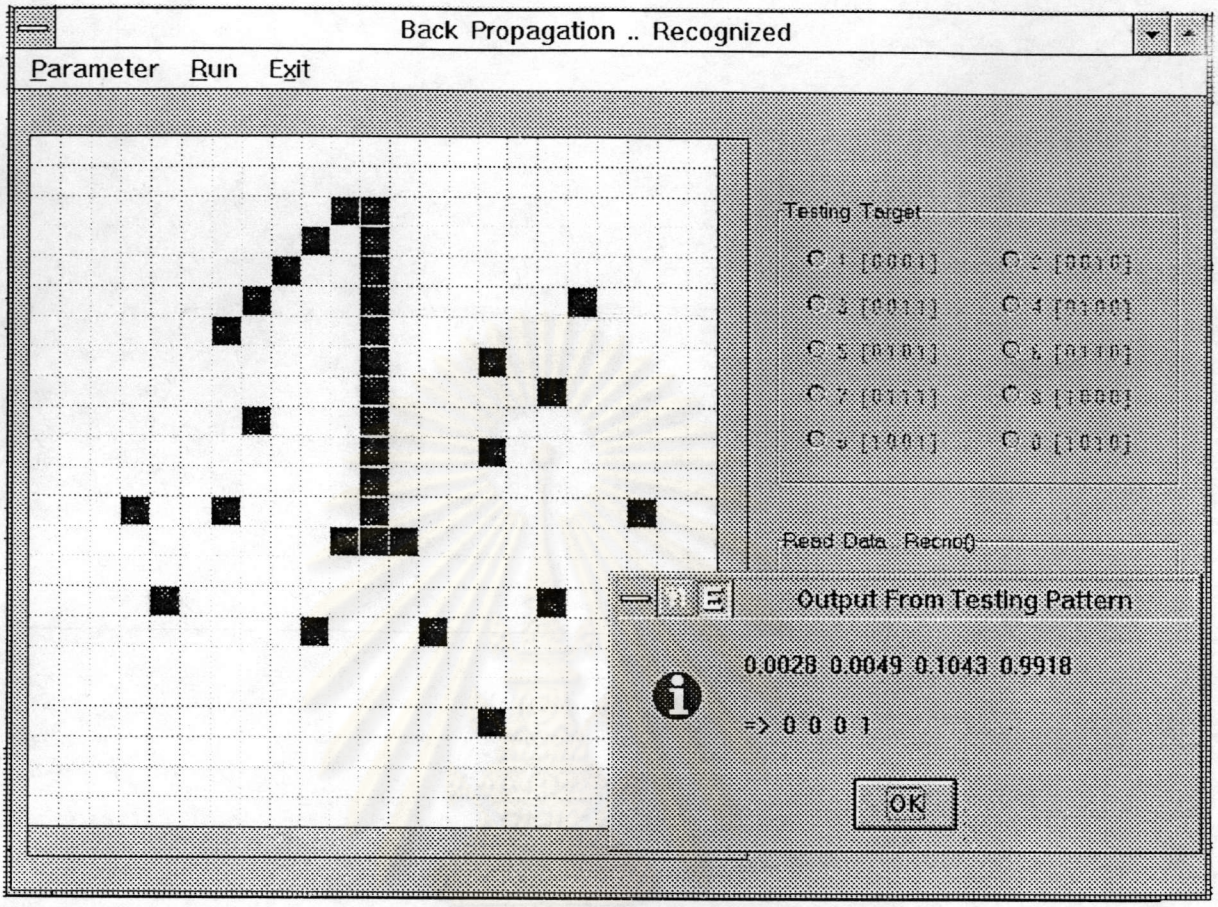
ในระหว่างการสอนจะแสดงค่าเฉลี่ยความผิดพลาดกำลังสองลงไปในลิสต์บอกซ์ และทำการบันทึกค่านี้ลงแฟ้มข้อมูล เพื่อให้สามารถนำมาแสดงในรูปกราฟเพื่อให้เห็นถึงความเคลื่อนไหวของค่าเฉลี่ยความผิดพลาดกำลังสองนี้ได้ ตามรูป 5.9



รูปที่ 5.9 แสดงกราฟของค่าเฉลี่ยความผิดพลาดกำลังสองในแต่ละรอบ

<u>การทดสอบการจดจำ</u>

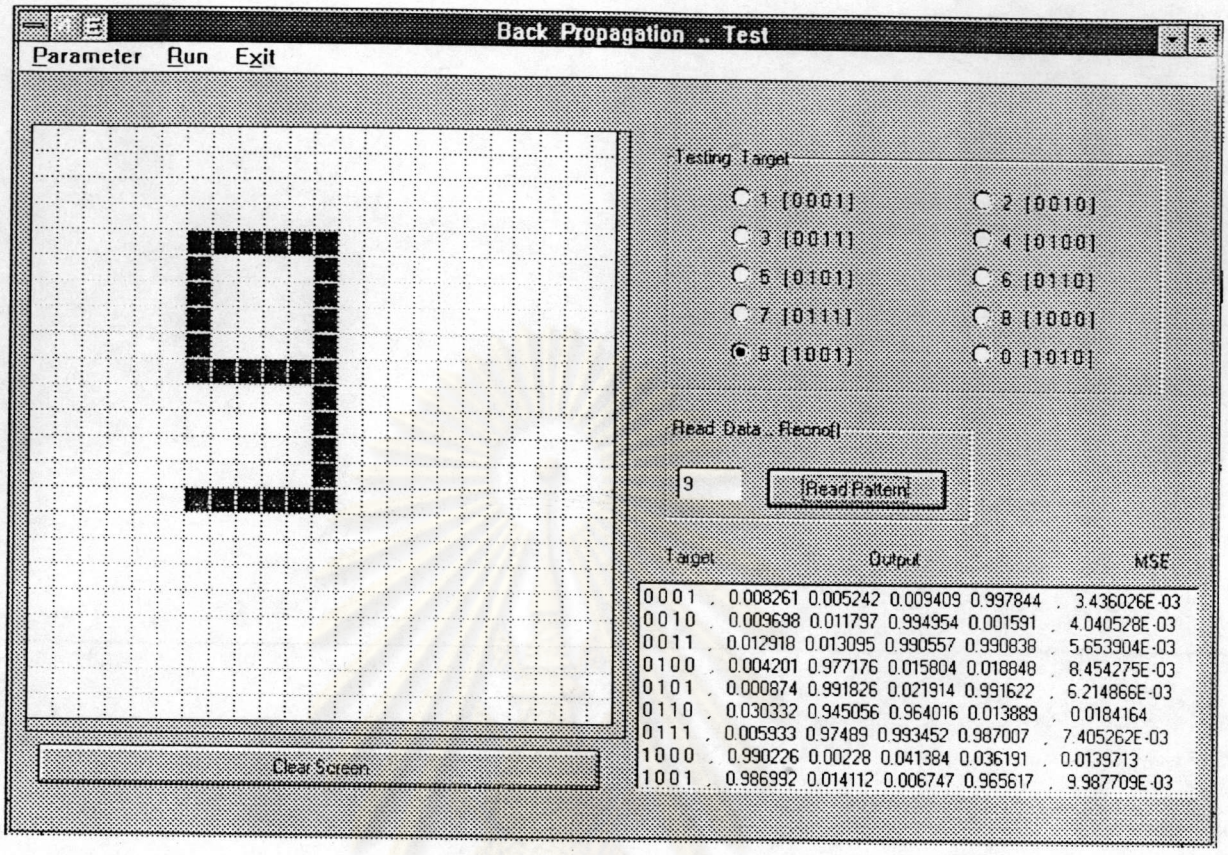
ในทำนองเดียวกับการสอนจะส่งข้อมูลจากกริด เป็นข้อมูลนำเข้าโดยไม่ต้องระบุว่าค่าเป้าหมายคืออะไร เมื่อได้ผลลัพธ์จะส่งกลับมายังวิชวลเบสิคที่เป็นโปรแกรมประยุกต์ใช้ และจะทำการวัดค่าผลลัพธ์ว่าคือค่าอะไร แล้วแสดงผลบนหน้าจอ ตามรูป 5.10 จะเพิ่มค่ารบกวนเข้าไปด้วย



รูปที่ 5.10 แสดงการทดสอบเลขสองโดยเพิ่มค่ารบกวน

<u>การทดสอบอื่น</u>

เพื่อให้ผู้ใช้สามารถรู้ได้ว่าค่าน้ำหนักจากการสอนนั้นทำให้ชุดข้อมูลใดสามารถเรียนรู้ได้ดีที่สุด โดยที่จะทำการคำนวณค่าความผิดพลาดกำลังสองของแต่ละโหนดของผลลัพธ์กับแต่ละโหนดของค่าเป้าหมาย แล้วแสดงในลิสต์บอกซ์ ตามรูป 5.11



รูปที่ 5.11 แสดงการทดสอบการจดจำ