類神經網路轉譯成 C++（作者：張藝瀚）

我的第一支類神經網路程式終於誕生啦!

雖然只是轉譯自別人的 C# 程式範例,雖然只是簡單的 XOR Gate,依我的理解補上缺的程式碼, 第一次執行成功看到輸出值逐漸收斂, 感覺很有成就感, 總算實現了多年的心願! 後續目標是做出更複雜的模型 (好歹要有反饋式), 做成 Multithread, 做到雲端...

感謝 C#.Net 版的原作者漠哥, 同意我轉譯成 C++ 使用, 原作網址是:

- http://mogerwu.pixnet.net/blog/post/25518602

轉譯好, 確定可以在 Linux 下編譯執行的C++程式碼分享如下:

```
/* ================================================================================
    Summary :
        類神經網路 - 學習機器人
    Compiler :
        linux:
            g++ -lrt -o ./Neural3 ./Neural3.cpp
    Usage :
        ./Neural3
    Reference :
       類神經網路-神經網路物件 @ 人生四十宅開始二號宅  痞客邦 PIXNET
         http://mogerwu.pixnet.net/blog/post/25518602
    Remark :
    History :
    yyyy.mm.dd Author            Discription
    ---------- ----------------- -------------------------------------------
    2010.11.10 Yihhann Chang     Translate from the C# source code of Moger Wu
    ---------- ----------------- -------------------------------------------
    Test:
        ./Neural3

================================================================================ */

#include <ctype.h>
#include <math.h>   // for exp(), fabs(), sqrt()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <time.h>   // for srand()

// 之前寫的類神經程式因為考慮到多執行緒，許多人看不懂，這一篇使用陣列來表達
// 神經元，並且完全不考慮多執行緒的問題，應該比較容易理解。

// 首先當然就是設計神經元，Value為神經元輸出值，如果神經元位於輸入層，它
// 同時也是輸入值，這樣設計是為了後面計算的程式好寫。神經元初始化的時候
// 必須要告訴他上一層的神經元個數，這樣他才能準備好神經鍵陣列Synapse。初
// 始化的時候讓閥值（GateValue)、神經鍵（Synapse）都使用亂數設定初始值，
// 根據經驗如果使用固定值，也就是1與-1交錯的初始值，在某些案例中可能不容
// 易收斂。
//
// 同時神經元也包含誤差值(diffentValue)、閥值修正(fixGateValue)、神經鍵
// 修正(fixSynapse[])，這樣把所有的值都放一起應該比較容易懂了吧。
class Element
{
    public :
    double Value;
    double GateValue;
    double *Synapse;

    // internal :
    double diffentValue;
    double fixGateValue;
    double *fixSynapse;
    int UpperLayerSize; // the length of Synapse and fixSynapse

    public :

    Element(int upperLayerSize) {
        int s;

        UpperLayerSize = upperLayerSize;
        Synapse = new double[UpperLayerSize];
        fixSynapse = new double[UpperLayerSize];

        if (UpperLayerSize > 0) {
            GateValue = ( (double)rand() / RAND_MAX ) * 2 - 1;
            for ( s = 0; s < UpperLayerSize; s++) {
                Synapse[s] = ( (double)rand() / RAND_MAX ) * 2 - 1;
            }
        }
    }
```

```
    // 解構式, 釋放動態配置的資源
    ~Element() {
        if( Synapse != NULL ) delete Synapse;
        if( fixSynapse != NULL ) delete fixSynapse;
    }

};
// end of class Element

// 接著解釋網路元件的設計。
//     Element ***Elements;
// 用一個二維的動態陣列來儲存神經元，陣列第一個註腳就是神經層，第二個註腳
// 就是每個神經層的神經元。

const int ELEMENTS_LENGTH = 3;   // Network::Elements.length, 三種 Layer
class Network {
    public:
    Element ***Elements;
    double *Standar;
    double DiffentValue;

    // Elements[?].length, 三種 Layer 的元素數, 方便跑迴圈用
    int Elements_lengths[ELEMENTS_LENGTH];

    // 為了程式方便，設計兩個屬性，直接傳回輸入層和輸出層。
    Element ** OutputLayer; // Elements[2];
    Element ** InputLayer;  // Elements[0];

    Network(int InputLayerSize, int HiddenLayerSize, int OutputLayerSize) {
        // 使用動態的方式宣告每一層的大小，這樣也比較符合實際網路架構。
        Elements = new Element **[3];
        Elements[0] = new Element *[InputLayerSize];
        Elements[1] = new Element *[HiddenLayerSize];
        Elements[2] = new Element *[OutputLayerSize];

        OutputLayer = Elements[2];
        InputLayer = Elements[0];
        Elements_lengths[0] = InputLayerSize;
        Elements_lengths[1] = HiddenLayerSize;
        Elements_lengths[2] = OutputLayerSize;

        Standar = new double[OutputLayerSize];

        int upperLayerSize = 0;
```

```
    for (int l = 0; l < ELEMENTS_LENGTH; l++) {
        for (int e = 0; e < Elements_lengths[l]; e++) {
            Elements[l][e] = new Element(upperLayerSize);
        }
        upperLayerSize = Elements_lengths[l];
    }
}
// end of Network(int InputLayerSize, int HiddenLayerSize, int OutputLayerSize)

// 解構式, 釋放動態配置的資源
~Network() {
    if( Elements != NULL ) {
        for (int l = 0; l < ELEMENTS_LENGTH; l++) {
            if( Elements[l] != NULL ) {
                for (int e = 0; e < Elements_lengths[l]; e++) {
                    if( Elements[l][e] != NULL ) delete Elements[l][e];
                }
                delete Elements[l];
            }
        }
        delete Elements;
    }
    if( Standar != NULL ) delete Standar;
}
// end of ~Network()


// 因為網路的修正動作是將所有的範例都計算過修正值之後，再一次進行修正，並
// 且再用新的網路進行計算，因此必須在計算之前將網路的動態值歸零，也就是說
// 下面這段程式裏面所歸零的值都是加總的值，必須在新的週期開始的時候清除掉。
void ClearValue() {
    DiffentValue = 0;
    for (int l = 0; l < ELEMENTS_LENGTH; l++) {
        for (int e = 0; e < Elements_lengths[l]; e++) {
            Elements[l][e]->Value = 0;
            Elements[l][e]->diffentValue = 0;
            Elements[l][e]->fixGateValue = 0;
            for (int s = 0; s < Elements[l][e]->UpperLayerSize; s++) {
                Elements[l][e]->fixSynapse[s] = 0;
            }
        }
    }
    // ==== DEBUG ==== TODO : LastHidden 不曉得是哪來的, 後面也沒用到
    // for (int h = 0; h < LastHidden.Length; h++) {
```

```
    //      LastHidden[h] = 0;
    // }
}

// 計算網路的輸出值，看過書的應該可以看得懂，數學公式實在不好貼，等我想到
// 好方法再補上吧。
void Summation() {
    for (int l = 1; l < ELEMENTS_LENGTH; l++) {
        for (int e = 0; e < Elements_lengths[l]; e++) {
            double outvalue = -Elements[l][e]->GateValue;
            for (int s = 0; s < Elements[l][e]->UpperLayerSize; s++) {
                outvalue += Elements[l - 1][s]->Value *
                    Elements[l][e]->Synapse[s];
            }
            Elements[l][e]->Value = (double)(1 / (1 + exp(-outvalue)));
        }
    }
}

// 計算網路誤差值，這裡我把公式分成兩種，一種是用來計算修正網路值用的，
// 使用書上所寫的公式。而另一個是給人看的，同時也是輸出值夠精密到可以跳
// 出的依據。為甚麼要這麼做請參考這一篇，雖然很多人來看，還是沒有人告訴
// 我答案，所以我用土方法解決這個問題。
void CalcDiffent() {
    //output layer
    for (int e = 0; e < Elements_lengths[2]; e++) {
        //給電腦看的用標準差公式
        OutputLayer[e]->diffentValue =
            (Standar[e] - OutputLayer[e]->Value) *
            (OutputLayer[e]->Value * (1 - OutputLayer[e]->Value) + 0.01);
        //給人看的用傳統公式
        DiffentValue += fabs(Standar[e] - OutputLayer[e]->Value);
    }
    //hidden layer
    for (int l = ELEMENTS_LENGTH - 2; l > 0; l--) {
        for (int e = 0; e < Elements_lengths[l]; e++) {
            double sumDiff = 0;
            for (int ne = 0; ne < Elements_lengths[l + 1]; ne++) {
                sumDiff += Elements[l + 1][ne]->Synapse[e] *
                    Elements[l + 1][ne]->diffentValue;
            }
            Elements[l][e]->diffentValue = (Elements[l][e]->Value *
                (1 - Elements[l][e]->Value)) * sumDiff;
        }
```

```
        }
    }

    // 得到誤差值之後就可以依照誤差值來得到修正值，因為要所有的範例都學習過
    // 才進行網路修正，所以所有的修正都是累加的。公式還是請自行參考書上說明
    // 。事實上這一段程式可以跟計算誤差值的程式合併，分開來寫比較容易看得懂
    // ，如果想要效能好一點就請將它合併在一起。
    void CalcFixValue(double LearnSpeed) {
        for (int l = ELEMENTS_LENGTH - 1; l > 0; l--) {
            for (int e = 0; e < Elements_lengths[l]; e++) {
                Elements[l][e]->fixGateValue =
                    -LearnSpeed * Elements[l][e]->diffentValue;
                for (int ue = 0; ue < Elements_lengths[l - 1]; ue++) {
                    Elements[l][e]->fixSynapse[ue] +=
                        LearnSpeed * Elements[l][e]->diffentValue *
                            Elements[l - 1][ue]->Value;
                }
            }
        }
    }

    // 當所有的範例都學習完成之後，當然就是要實際修正網路內容，修正值取平均，
    // 所以需要傳入範例個數來進行運算。
    void FixNetwork(int SampleCount) {
        for (int l = 1; l < ELEMENTS_LENGTH; l++) {
            for (int e = 0; e < Elements_lengths[l]; e++) {
                Elements[l][e]->GateValue +=
                    Elements[l][e]->fixGateValue / sqrt((double)SampleCount);
                for (int s = 0; s < Elements[l][e]->UpperLayerSize; s++) {
                    Elements[l][e]->Synapse[s] +=
                        Elements[l][e]->fixSynapse[s] /
                            sqrt((double)SampleCount);
                }
            }
        }
    }
};
// end of class Network

// 學習機器人基礎類別
// 這個類是個基礎類，因為載入資料和將資料放進輸入層的動作，每一個案例都
// 不相同，因此使用必須繼承的關鍵字abstract宣告這個類別，並且宣告兩個方
// 法LoadSample和LoadData為必須實做的。
class RobotBase {
```

```
public :
int SampleCount;

// internal :
int inputLayerSize, outputLayerSize;
Network *worknet;
int noBestCount;
int learnSamples;
double bestDiffent; // = 10000;

// 建構式，賦與初值
RobotBase(){
    bestDiffent = 10000;
    worknet = NULL;
}

// 解構式, 釋放動態配置的資源
~RobotBase(){
    if( worknet != NULL ) delete worknet;
}

// 純虛擬函式, 由繼承者實作
virtual void LoadSample() = 0;
virtual void LoadData(int SampleNo) = 0;

// public delegate void OnCycleFinish(int CycleNo, double BestDiffent, double NewDiffent);
// public event OnCycleFinish EventCycleFinish;

// public delegate void OnBadLearning(int NoBestCount);
// public event OnBadLearning EventBadLearning;

// 最後就是最重要的學習過程了，基本程式和前面VB.Net的寫法一樣，就是用
// 【找不到最佳值的次數】或【達到預定的精密度】來決定學習是否完成。不
// 過有的時候誤差值調整的幅度很小，可能導致跑了幾十萬次都還出不來，可
// 以考慮再增加一個對cycle的限制。
virtual void Learning(double LearnSpeed, int HiddenLayerSize,
    int NoBestLimit, double Precision)
{
    worknet = new Network(inputLayerSize, HiddenLayerSize, outputLayerSize);
    bestDiffent = 10000;
    int cycle = 0;
    noBestCount = 0;
    while ((noBestCount < NoBestLimit) && (bestDiffent > Precision)) {
        double newDiffent;
```

```
            cycle++;
            worknet->ClearValue();
            for (int sampleNo = 0; sampleNo < learnSamples; sampleNo++) {
                LoadData(sampleNo);
                worknet->Summation();
                worknet->CalcDiffent();
                worknet->CalcFixValue(LearnSpeed);
                // ==== DEBUG ====
                int e;
                printf( "Input:( " );
                for( e = 0; e < worknet->Elements_lengths[0]; e++)
                    printf( "%+f ", worknet->InputLayer[e]->Value );
                printf( "), Standard:( " );
                for( e = 0; e < worknet->Elements_lengths[2]; e++)
                    printf( "%f ", worknet->Standar[e] );
                printf( "), Output:( " );
                for( e = 0; e < worknet->Elements_lengths[2]; e++)
                    printf( "%f ", worknet->OutputLayer[e]->Value );
                printf( ")\n" );
            }
            newDiffent = worknet->DiffentValue / learnSamples;
            if (newDiffent < bestDiffent) {
                bestDiffent = newDiffent;
                noBestCount = 0;
            }
            else {
                noBestCount++;
            }
            // if (EventBadLearning != null) EventBadLearning(noBestCount);
            worknet->FixNetwork(learnSamples);
            // if (EventCycleFinish != null) EventCycleFinish(cycle, bestDiffent, newDiffent);

            // ==== DEBUG ====
            printf( "cycle=%d, noBestCount=%d, bestDiffent=%0.8f, newDiffent=%0.8f\n",
                cycle, noBestCount, bestDiffent, newDiffent );
        }

        delete worknet;
        worknet = NULL;
    }
};
// end of class RobotBase
```

```
// XOR為範例
class RobotXOR : public RobotBase
{
    public :
    // XOR 學習機
    RobotXOR() {
        inputLayerSize = 2;      // 輸入值 2 個
        outputLayerSize = 1;     // 輸出結果1 個
        learnSamples = 4;        // 輸入組合只有 4 種
    }

    // 設定輸入樣本, 及標準答案
    void LoadData( int sampleNo )
    {
        switch ( sampleNo )
        {
            case 0:
                worknet->InputLayer[0]->Value = -1;
                worknet->InputLayer[1]->Value = -1;
                worknet->Standar[0]           =  0;
                break;
            case 1:
                worknet->InputLayer[0]->Value = -1;
                worknet->InputLayer[1]->Value =  1;
                worknet->Standar[0]           =  1;
                break;
            case 2:
                worknet->InputLayer[0]->Value =  1;
                worknet->InputLayer[1]->Value = -1;
                worknet->Standar[0]           =  1;
                break;
            case 3:
                worknet->InputLayer[0]->Value =  1;
                worknet->InputLayer[1]->Value =  1;
                worknet->Standar[0]           =  0;
                break;
        }
    }
    // end of void LoadData( int sampleNo )

    // 這函數應該是用來取代 LoadData, 當樣本數量龐大時, 可以改從檔案/資料庫載入
    void LoadSample()
    {
    }
```

```
    // 通常隱藏層的寬度可以設置為
    // hiddenLayerSize=(inputLayerSize+outputLayerSize)/2
    // ，但是許多案例並不能滿足需求，而是要用嘗試錯誤法去求得合適的隱藏層寬度
    // 。前面Learning宣告為可以被重新包裝的，就是?了這個，當然你也可以把它包
    // 裝在一起。如果一個學習週期跳出後，精密度不夠就試著調整隱藏層寬度。
    void Learning(double LearnSpeed, int HiddenLayerSize, int CycleLimit,
        double Precision)
    {
        while (bestDiffent > Precision) {
            // if (EventHiddenLayerChange != null) EventHiddenLayerChange(HiddenLayerSize);
            this->RobotBase::Learning(LearnSpeed, HiddenLayerSize, CycleLimit,
                Precision);
            if (bestDiffent > Precision) {
                HiddenLayerSize = (int)(HiddenLayerSize * 1.2 + 1);
            }
            // ==== DEBUG ====
            printf( "bestDiffent=%0.8f, Precision=%0.8f, HiddenLayerSize=%d\n\n",
                bestDiffent, Precision, HiddenLayerSize );
        }
    }
};
// end of class RobotXOR


// ========================================================================
int main()
{
    RobotXOR *Robot;
    srand ( time(NULL) );   /* initialize random seed: */
    printf( "Hello~\n" );

    Robot = new RobotXOR;

    Robot->Learning(
        1000,     // double LearnSpeed
        10,       // int HiddenLayerSize
        10,       // int CycleLimit
        0.00001   // double Precision
    );
    delete Robot;

    printf( "Bye~\n" );
}
```