國立台灣海洋大學資訊工程系 2A　C++ 程式設計 期中考參考答案

姓名：_____

系級：_____

學號：_____

105/04/19 (二)

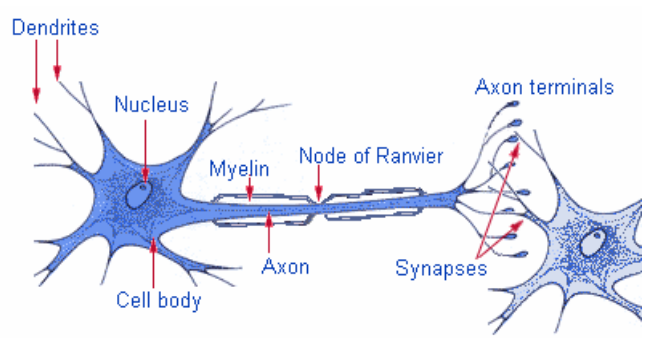
考試時間：**09:30 – 12:00**

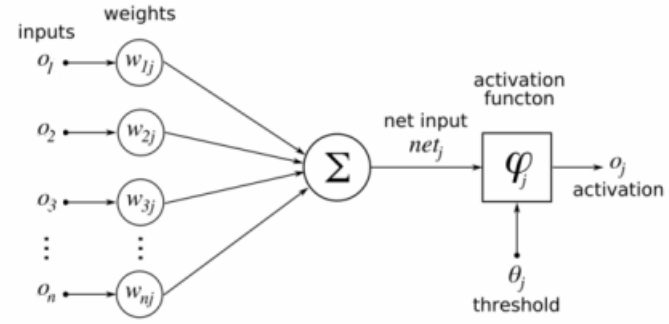
請儘量回答，總分有 **125**，看清楚每一題的分數再回答

考試規則：1. 不可以翻閱參考書、作業及程式
2. 不可以使用任何形式的電腦 (包含手機、計算機、相機以及其它可運算或是連線的電子器材)
3. 請勿左顧右盼、請勿交談、請勿交換任何資料、試卷題目有任何疑問請舉手發問 (看不懂題目不見得是你的問題，有可能是中英文名詞的問題)、最重要的是隔壁的答案可能比你的還差，白卷通常比錯得和隔壁一模一樣要好
4. 提早繳卷同學請直接離開教室，請勿逗留喧嘩
5. 違反上述任何一點之同學一律依照學校規定處理
6. 繳卷時請繳交***簽名過之試題卷及答案卷***

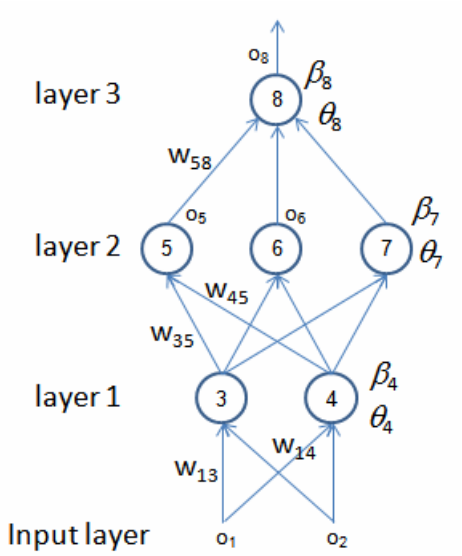
撰寫程式的能力隨著你對於資料以及處理方法***抽象化***的程度慢慢增加而提昇，回想你剛學習怎麼寫**函式**的時候，可能覺得搞資訊的人真是莫名其妙，明明可以都寫在 main()裡面的為什麼要那麼麻煩寫到各個函式裡，有一種衝動想要都寫在同一個函式裡；剛接觸**堆疊**資料結構的時候，覺得不過是一個陣列而已，有一種衝動想要直接修改堆疊資料裡的最上面一個元素、第二個元素、或是任意一個元素，有點難接受一定要透過 push()/pop()來存取堆疊的要求；凡事都看到或管到最底層當然可以把事情弄得很清楚，可是也因為每個人腦袋裡能夠處理的事情複雜度有限，一旦分心處理細節，就沒有辦法正確快速地完成更大規模的程式…想像一下如果今天你控制機器時都要用 CPU 的機器指令一列一列地完成的話，你能寫出多大的程式，你運用 C 語言撰寫程式時已經接受了很多底層的抽象化概念，例如 printf()就能夠正確地把給它的資料用指定的格式顯示在螢幕上，運用這樣的抽象化，你的心思不要再煩惱 printf()函式裡面的細節，不要管到最底層輸入輸出的動作，可以專心完成需要你設計的工作。你要練習運用已經完成的元件來製作新的功能，要快速掌握各個元件所保證的功能，不要衝動去拆開它，當然也慢慢地習慣設計出一個一個的元件。在**物件導向的程式設計**中，最基本但是也最讓許多人在學習時迷失重點的就是**物件**，設計物件把底層的細節**封裝**起來，組合各種物件現有的功能來完成更大規模的程式功能，練習分工合作，該交付給別人負責的，不要因為莫名的衝動而弄不清楚權責，打破物件的封裝。這次考試裡我們著重的就在於如何完成基本物件的設計。

前一陣子 Google DeepMind 的 AlphaGo 相當風光，它是一種深度學習(deep learning)的類神經網路(artificial neural network)，最基本的概念由下圖左的神經細胞構成，一個神經細胞有很多的樹突(Dendrites)來接受刺激，神經細胞如果被激發了，就會沿著神經軸突(Axon)傳送訊號，透過神經突觸(Synapse)將訊號傳導到下一個神經細胞，





上圖右是一個神經細胞的模型，輸入為 o_i，神經細胞計算並且輸出 $o_j=\varphi_j(\sum_i w_{ij}o_i-\theta_j)$，其中激發函式(activation function) $\varphi_j(net_j)$ 模擬神經元細胞的反應，例如 step function: $o_j=\varphi_j(net_j)=\begin{cases}1, & net_j\geq 0\\ 0, & net_j<0\end{cases}$，linear combination function: $\varphi_j(net_j)=\sum_i w_{ij}o_i-\theta_j$，或是 sigmoid function: $o_j=\varphi_j(net_j)=\frac{1}{1+e^{-\beta_j net_j}}$。下圖左是一個前向連結的多層感知器(multilayer perceptron)，包含三層神經細胞，其中標示 j 的神經細胞的參數 w_{ij},β_j,θ_j 是經由不斷地學習調整出來的，一個深度學習的類神經網路可能包含數十層的神經細胞，以下我們設計**神經細胞 Neuron** 類別，並且運用它來設計**類神經網路類別 NeuralNet**：請根據下圖右的檔案內容以及下列問題，依序來設計這兩個類別，請分別註明程式應該放在哪一個檔案中，以及需要引入的標頭檔案



net.dat	
3	三層網路
2	兩個輸入 o_1, o_2
2	第一層有兩個神經細胞
3 1.1 0.5	細胞 3 的參數 $\beta_3=1.1$，$\theta_3=0.5$
2	細胞 3 有兩個輸入
-0.1 1	細胞 3 連接第一個輸入 o_1 的係數 w_{13}
0.3 2	細胞 3 連接第二個輸入 o_2 的係數 w_{23}
4 1.1 0.3	細胞 4 的 β_4，θ_4
2	細胞 4 有兩個輸入
-0.15 1	細胞 4 連接第一個輸入 o_1 的係數 w_{14}
0.4 2	細胞 4 連接第二個輸入 o_2 的係數 w_{24}
3	第二層有三個神經細胞
…	
1	第三層有一個神經細胞
8 1.1 -0.2	細胞 8 的參數 β_3，θ_3
3	細胞 8 有三個輸入
0.14 5	細胞 8 連接第一個輸入 o_5 的係數 w_{58}
-0.1 6	細胞 8 連接第二個輸入 o_6 的係數 w_{68}
0.4 7	細胞 8 連接第三個輸入 o_7 的係數 w_{78}

1. [10] 一個 Neuron 物件內應該要管理它所有的參數，包括 id, w_{ij},β_j,θ_j，每一個神經細胞的輸入並沒有限制個數，需要動態地配置 double 型態的陣列紀錄 w_{ij}，w_{ij} 和其來源的神經細胞有關，因此也要紀錄是由哪些細胞的輸出接過來的，需要動態地配置可以紀錄 Neuron 指標的陣列來完成，最後每一個 Neuron 物件都有一個單一的輸出值，請設計 Neuron 類別的資料成員

Sol:

```
// neuron.h
#include <fstream>
#include <map>
using namespace std;

class Neuron
{
public:
    Neuron(int id);
    Neuron(double beta, double theta, double weights[], int size);
    Neuron(istream &is, map<int, Neuron*> &neuronPool);
    Neuron(const Neuron &src);
    ~Neuron();
    Neuron &operator=(const Neuron &rhs);
    void calculate();
```

```
        double getOutput() const;
        void saveFile(ostream &os) const;
        void setOutput(double x);
    private:
        double m_beta;
        double m_theta;
        double *m_weights;
        Neuron **m_dendrites;
        double m_value;
        int       m_inputSize;
        int       m_id;
};
```

2. [5] 對於整個網路來說輸入層中每個輸入只是一個數值，但是為了簡化設計，我們打算把輸入看成一個 Neuron 類別的物件的輸出，請配合設計一個 Neuron 類別的建構元函式(ctor)，傳入一個整數 id 值，運用初始化串列(initialization list)將所有參數設為 0，這個建構元在讀取 net.dat 裡面指定整個網路輸入個數時使用到

Sol:

```
// neuron.cpp
Neuron::Neuron(int id)
    : m_beta(0.), m_theta(0.), m_weights(NULL),
      m_dendrites(NULL), m_value(0), m_inputSize(0), m_id(id)
{
}
```

3. [15] 請設計 Neuron 類別的建構元函式(ctor)，由串流中讀取一個神經細胞的參數資料來建構物件，請以 new[] 來配置存放 w_{ij} 和來源 Neuron 物件指標的記憶體，Neuron 物件指標代表連接到這個物件的輸入的那些神經細胞 (稍後配合 NeuralNet 的建構元的撰寫，請運用 map<int, Neuron*> 物件來建立整個網路的架構)

Sol:

```
// neuron.cpp
Neuron::Neuron(istream &is, map<int, Neuron*> &neuronPool)
{
    int src_neuron_id;
    is >> m_id >> m_beta >> m_theta >> m_inputSize;
    m_weights = new double[m_inputSize];
    m_dendrites = new Neuron*[m_inputSize];
    for (int i=0; i<m_inputSize; i++)
    {
        is >> m_weights[i];
        is >> src_neuron_id;
        m_dendrites[i] = neuronPool[src_neuron_id];
    }
    neuronPool[m_id] = this;
}
```

4. [5] 配合上題中動態配置記憶體的機制，請撰寫 Neuron 類別的解構元函式(dtor)釋放所配置的記憶體

Sol:

```
// neuron.cpp
Neuron::~Neuron()
{
    delete[] m_weights;
    delete[] m_dendrites;
}
```

5. [10] 請撰寫 Neuron 類別中計算一個神經細胞輸出的成員函式 void calculate()，實作 $net_j = \sum_i w_{ij} o_i - \theta_j$ ，

$o_j = \varphi_j(net_j) = \frac{1}{1+e^{-\beta_j net_j}}$ 的機制，把計算出來的輸出值 o_j 記錄在資料成員中

Sol:

```
// neuron.cpp
#include <cmath>
using namespace std ;
void Neuron::calculate()
{
    double net = -m_theta;
    for (int i=0; i<m_inputSize; i++)
        net += m_weights[i] * m_dendrites[i]->m_value;
    m_value = 1/(1+exp(-m_beta*net));
}
```

6.［10］請撰寫 Neuron 類別的拷貝建構元

Sol:

```
// neuron.cpp
Neuron::Neuron(const Neuron &src)
    : m_beta(src.m_beta), m_theta(src.m_theta), m_weights(NULL),
      m_dendrites(NULL), m_value(src.m_value),
      m_inputSize(src.m_inputSize), m_id(src.m_id)
{
    m_weights = new double[m_inputSize];
    m_dendrites = new Neuron*[m_inputSize];
    for (int i=0; i<m_inputSize; i++)
        m_weights[i] = src.m_weights[i],
        m_dendrites[i] = src.m_dendrites[i];
}
```

7. [10] 請撰寫 Neuron 類別的設定運算子(assignment operator)

Sol:

```
// neuron.cpp
Neuron& Neuron::operator=(const Neuron &rhs)
{
    if (&rhs==this) return *this;
    delete[] m_weights;
    delete[] m_dendrites;
    m_beta = rhs.m_beta;
    m_theta = rhs.m_theta;
    m_inputSize = rhs.m_inputSize;
    m_weights = new double[m_inputSize];
    m_dendrites = new Neuron*[m_inputSize];
    for (int i=0; i<m_inputSize; i++)
        m_weights[i] = src.m_weights[i],
        m_dendrites[i] = rhs.m_dendrites[i];
    return *this;
}
```

8. [10] 請撰寫 Neuron 類別將本身資料寫到輸出串流的 saveFile(ostream &os) const 成員函式(請依照 net.dat 格式)

Sol:

```
// neuron.cpp
void Neuron::saveFile(ostream &os) const
```

```
{
    os << m_id << ' ' << m_beta << ' ' << m_theta << endl;
    os << m_inputSize << endl;
    for (int i=0; i<m_inputSize; i++)
        os << m_weights[i] << ' ' << m_dendrites[i]->m_id << endl;
}
```

9. [10] 請設計 NeuralNet 類別的資料成員，為了一層一層地計算每一個神經細胞的輸出，需要完整紀錄每一層有哪些 Neuron 物件所構成，請以 vector<Neuron*>來紀錄每一層的 Neuron 物件，因為層數不確定，所以也運用 vector 來存放前面這個 vector<Neuron*>物件的指標

Sol:

```
// neuralnet.h
#include <vector>
using namespace std;
class Neuron;

class NeuralNet
{
public:
    NeuralNet(istream &is);
    ~NeuralNet();
    void saveFile(ostream &os) const;
    void calculate(double inputs[]);
    vector<double> getOutput() const;
private:
    NeuralNet(const NeuralNet &src);
    NeuralNet &operator=(const NeuralNet &rhs);
    vector<vector<Neuron *>*> m_layers;
};
```

10. [10] 請設計 NeuralNet 類別的建構元函式(ctor)，由串流中讀取資料來建構整個類神經網路，函式內請運用一個 map<int, Neuron*> 物件來紀錄每一個建立起來的 Neuron 物件的 ID 值以及其指標，請修改 Neuron 類別由串流建構的 ctor，把這個物件的參考傳遞進去，如此在 Neuron 建構元中可以很快地運用 Neuron 指標建立出整個類神經網路 (請注意 Neuron 建構元中也需要維護這個 map 物件的內容)

Sol:

```
// neuralnet.cpp
NeuralNet::NeuralNet(istream &is)
{
    map<int, Neuron*> neuronPool;
    int nLayers, nInputs, nNeurons, i, j;
    is >> nLayers;
    is >> nInputs;
    m_layers.push_back(new vector<Neuron*>);
    for (i=0; i<nInputs; i++)
    {
        m_layers[0]->push_back(new Neuron(i+1));
        neuronPool[i+1] = (*m_layers[0])[i];
    }

    for (i=1; i<=nLayers; i++)
    {
        m_layers.push_back(new vector<Neuron*>);
        is >> nNeurons;
        for (j=0; j<nNeurons; j++)
            m_layers[i]->push_back(new Neuron(is, neuronPool));
```

```
    }
}
```

11. [5] 請設計一個成員函式 void NeuralNet::calculate(double inputs[])，一層一層依序計算由輸入層到輸出層所有 Neuron 物件的輸出，其中 inputs[]陣列內的數值需要設定到輸入層 Neuron 物件中，請額外設計 void Neuron::setOutput(double)介面來輔助

Sol:

```
// neuralnet.cpp
void NeuralNet::calculate(double inputs[])
{
    int i, j;
    for (j=0; j<m_layers[0]->size(); j++)
        (*m_layers[0])[j]->setOutput(inputs[j]);
    for (i=1; i<m_layers.size(); i++)
        for (j=0; j<m_layers[i]->size(); j++)
            (*m_layers[i])[j]->calculate();
}
// neuron.cpp
void Neuron::setOutput(double x)
{
    m_value = x;
}
```

12. [5] 請設計一個讀取輸出的介面函式 NeuralNet::getOutput() const，回傳 vector<double>型態的物件代表整個 NeuralNet 的輸出，最後一層所有的 Neuron 物件的輸出值向量就是整個 NeuralNet 物件的輸出，過程中請撰寫一個 double Neuron::getOutput() const 成員函式來取得個別 Neuron 的輸出

Sol:

```
// neuron.cpp
double Neuron::getOutput() const
{
    return m_value;
}
// neuralnet.cpp
vector<double> NeuralNet::getOutput() const
{
    vector<double> output;
    int nLayers = m_layers.size()-1;
    for (int i=0; i<m_layers[nLayers]->size(); i++)
        output.push_back((*m_layers[nLayers])[i]->getOutput());
    return output;
}
```

13. [10] 請撰寫 NeuralNet 類別將本身資料寫到輸出串流的 saveFile(ostream &os) const 成員函式(請依照 net.dat 格式)

Sol:

```
// neuralnet.cpp
void NeuralNet::saveFile(ostream &os) const
{
    int i, j;
    int nLayers = m_layers.size()-1;
    int nNeurons;
    os << nLayers-1 << endl;
    os << m_layers[0]->size() << endl;

    for (i=1; i<=nLayers; i++)
    {
```

```
            nNeurons = m_layers[i]->size();
            os << nNeurons << endl;
            for (j=0; j<nNeurons; j++)
                (*m_layers[i])[j]->saveFile(os);
        }
    }
    下面重寫一次換成用 iterator 來做
    void NeuralNet::saveFile(ostream &os) const
    {
        os << m_layers.size()-1 << endl;
        os << (*m_layers.begin())->size() << endl;
        vector<vector<Neuron*>*>::const_iterator i; // because this is a const function
        for (i=m_layers.begin()+1; i!=m_layers.end(); ++i)
        {
            os << (*i)->size() << endl;
            for (vector<Neuron*>::iterator j=(*i)->begin(); j!=(*i)->end(); ++j)
                (*j)->saveFile(os);
        }
    }
```

14. [5] 請撰寫 NeuralNet 類別的解構元函式

Sol:

```
// neuralnet.cpp
NeuralNet::~NeuralNet()
{
    int i, j;
    for (i=m_layers.size()-1; i>=0; i--)
    {
        for (j=0; j<m_layers[i]->size(); j++)
            delete (*m_layers[i])[j];
        delete m_layers[i];
    }
}
```

請留心有參考關係的物件解構的順序和建構時顛倒以維持程式在多執行緒時的正確性，也就是說任何時候要維持每一個物件不要參考到已經解構掉的東西，因為第 i 層 Neuron 會參考到第 i-1 層 Neuron, 所以在第 i-1 層解構前一定要先解構第 i 層的 Neuron, 下面換成用 iterator 來做

```
NeuralNet::~NeuralNet()
{
    for (vector<vector<Neuron*>*>::reverse_iterator i=m_layers.rbegin(); i!=m_layers.rend(); ++i)
    {
        for (vector<Neuron*>::iterator j=(*i)->begin(); j!=(*i)->end(); ++j)
            delete *j;
        delete *i;
    }
}
```

請注意 rbegin()指的是容器裡的最後一個元素, rend()則是第一個元素之前的元素, operator++是由容器結束的地方往容器開頭的地方移動, vector<xxx>::iterator:operator->() 不是 smart pointer

15. [5] 請問如果 NeuralNet 類別不允許其客戶程式進行設定(assignment)的運算也不允許其客戶程式進行拷貝，該如何宣告?

Sol:

將拷貝建構元以及設定運算子定義為 private, 使得其它模組無法直接存取

```
// neuralnet.h
class NeuralNet
{
public:
    …
private:
```

```
    NeuralNet(const NeuralNet &src);
    NeuralNet &operator=(const NeuralNet &rhs);
    …
};
```

基本測試程式如下

```
// main.cpp
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream ifs("net.dat");
    NeuralNet nn(ifs);
    nn.saveFile(cout);

    double inputs[] = {1.5, 0.3};
    nn.calculate(inputs);
    vector<double> result = nn.getOutput();
    for (int i=0; i<result.size(); i++)
        cout << i << ':' << result[i] << endl;

    system("pause");
    return 0;
}
```