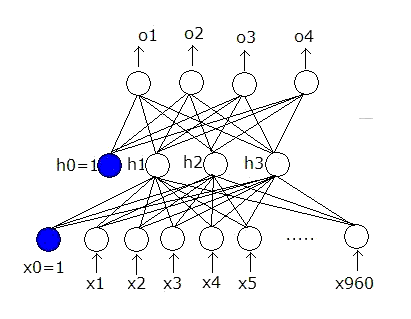
http://www.educity.cn/blog/user1/261/archives/2010/43657.html

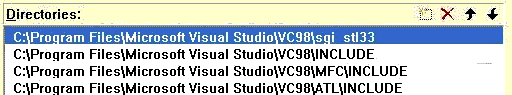
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | | |  | | --- | |  | | | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | |  | | --- | |  | | **BP神经网络实现人脸识别（一）(附图,附源码下载)** | | [ 2010-6-4 14:32:00 | By: c\_gao ] |  |  | | --- | |  |  |  | | --- | | **14**  [推荐](javascript:void(null))  由于单篇博客容量限制，被迫拆成几篇发布，见谅！  **欢迎转载，但请注明出处，谢谢！**     为了配合PPT讲解BP神经网络原理的需要,这几天写了这个基于BP神经网络的演示程序,主界面如下图，源码及演示程序下载见最下方（用VC6+WTL+SGI-STL实现）:  关于BP神经网络的基本原理,我觉得没有太多的必要在这里讲得很细,如果读者不是很清楚,可以先看看相关书籍或论文,可以推荐几个如:             **1.**<<Machine Learning>>的第4章,Tom.M.Mitchell著,此书非常好,把BP原理讲得非常透彻,由其是误差的公式推导,比别的书都说得清楚一点,该书出过中文版,由曾华军等人翻译.             **2.**<<Explorations in Parallel Distributed Processing, A Handbook of Models, Programs, and Exercises >>,这本手册第5章就是历史上第一次提出将BP算法应用于神经网络的,非常值得一看.网上能下到最新版(第二版的DRAFT版)的PDF,时间是2010年2月份的,很新了.             **3.**韩力群教授写的几本神经网络的书,都很不错,讲得比较浅显易懂             **4.**<<Neural Networks A Systematic Introduction>>,这本书的第7章讲BP,讲得很细,值得看,电子版网上能下到.             **5.**网上的教程也不少,比如这个: http://tech-algorithm.com/articles/backpropagation-neural-network/             **6.**实在理解不了,试着联系我吧,我也乐意与您沟通,我的联系方式在这个演示程序的About对话框内能找到.       在这里主要提几点:      **1.**BP神经网络属于监督学习范畴,除了给定训练样本数据外,必须给出该样本对应的标记      **2.**学习率和冲量项设置有一定讲究,不能随意,大了可能不收敛,小了收敛慢      **3.**隐藏层节点个数需要多少,没有理论指导,都是经验指导      **4.**是采用标准梯度下降,还是delta法则,关系不大,只要学习率足够小,delta法则可逼近标准梯度下降法.      **5.**BP神经网络的性能和样本训练前的前期处理有密切关系,识别率低,不一定是网络不好,可能是特征提取不好,特征组合有问题,等等,很多原因.        下面讲一下,怎么运行这个演示程序:      **1.**下载本文附带的压缩包,并解压至某路径,比如:  D:\      **2.**注册控件,可以双击那个reg\_ax.bat文件即可,它会自动注册同一路径下的facerecognization.dll文件,这是我写的用于显 示PGM图片的ActiveX控件,不注册的话,就不能显示图片了,显示效果如上面的截图(所有图片分辩率为32\*30,所以有点小),注册后就不要移动 这个dll文件,  否则需要再次注册(明白COM组件的工作原理就会知道了)      **3.**通过1,2两步,现在就可以正常运行BPNNTest.exe这个程序了.运行后界面如上图,不过这时还没有训练神经网络.       下载人脸库:      BP网络需要先训练,我写程序时使用的是CMU的人脸库,下载地址是:http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-8/faceimages/faces\_4.tar.Z 人脸图片用的是32\*30分辨率,PGM格式的图片,文件名命名规则是:人名\_人脸朝向\_心情\_是否戴太阳镜\_4.pgm,例如:danieln\_left\_angry\_open\_4.pgm,因此,从这个文件名中,我们可以得到四个信息,分别是:人名,朝向,心情,是否戴太阳 镜,而文件本身的数据可以作为训练网络的特征向量,而这四个信息则均可以用于特征向量对应的标记.        将上面链接下载的人脸库解压至某一路径下,如D盘根目录下,会发现faces\_4文件夹内有很多子文件夹,分别名为:an2i,at33...等,而这些子文件夹内就是各自的pgm图片文件.        因为,BP神经网络需要训练和测试两个样本集,你可以随便把faces\_4中的某个(或某几个)文件夹从中剪切出来,作为测试样本集,而其余的作为训练样本集.比如,将tammo文件夹从中剪切到D:\下.最终形成的目录结构应该类似于这样:  **训练样本集:   D:\faces\_4\很多子文件夹\很多pgm文件                       测试样本集:   D:\tammo\很多pgm文件**      注意faces\_4内都是目录,而tammo内都是pgm文件,这很重要,因为我现在程序写得就是按这个方式处理的.      准备工作就续,现在就可以训练和测试网络了:     **1.选择训练集** 点击对话框左上角的Select按钮,弹出对话框,先择之前解压后的faces\_4文件夹（由于博客容量限制，图略):   选中faces\_4文件夹后,点击OK按钮(你的电脑可能是"确定"按钮),程序会读取文件夹内所含PGM图片信息,并显示读到的第一张图片.      **2.训练BP网络**         **2.1** 你可以选择训练BP网络去学习什么功能,默认是学习Direction,即人脸朝向,你可以通过对话框左上角的ComboBox选择四种类型:人名,朝向,心情,是否戴太阳镜          **2.2** 选择迭代次数(iterate time)或停止迭代误差(Stop Error),两者默认值为100000和0.003,即分别表示迭代到100000次时停止(不管误差多少),或迭代直到误差小于指定值时停止(可能会 迭代很多很多次).这里说明一下,程序里,我为了让每张人脸图片都被训练到相同次数,使得这里的100000次是训练的下界,到达100000次迭代时, 可能还没有训练到样本集中最后一个样本,因此会多训练几次,如最上面的截图中,一共迭代了100215次.另外,由于程序使用的是delta法则来训练本样,这里的误差是每个样本训练完后的平均误差,而不是当前训练图片的误差.          **2.3** 确定隐藏层神经元个数(Hidden layer units),默认为3个(网络只含一个隐藏层,多层网络其实也一样),由于所需隐单元个数的多少目前仍而理论依据,因此当隐层个数太少,无法达到识别率时,考虑调大一点,不过由于隐层个数调大后,权值会增加,训练会明显变慢.另外,当BP网络要学习目标比较复杂时,隐层单元个数不因过少,不然可能无法收敛.          **2.3\*** 为什么不用设置输入层和输出层单元个数? 因为,为了方便,程序自动读取图片,分析文件的分辩率自动确定输入层单元个数(没有做特征提取,就是图片像素数),通过分析样本集中目标值的可取个数,自动确定输出层单元个数,比如,分析样本集文件后,得到所有Direction可取的值就只有4种,于是,如果采用"1of n"方式设计输出层单元(本程序采用的方式),就有4个输出单元.          **2.4** 确定学习速率(learning rate)和冲量(momentum),默认值都是0.3,我没尝试过很多值,大概设了一下,不要设太大,可能导致无法收敛,太小会收敛很慢,是很慢很慢,不信你把两个值都调成0.03,迭代100000次后的误差仍然很大.          **2.5** 参数都设好了,点击"Train"按钮 开始训练吧,对话框下方的"Status"会动态显示当前迭代的次数,而图片下方也会动态更新当前迭代中的图片路径.要说明的是:如果迭代次数少,很快就 会训练完,如果次数多的话,就看你电脑的性能了,在我的老本本(T40p)上,迭代100000次,大概是90几秒,现在的电脑速度都很快,训练时间应该 会更短.虽然我没有训练到将误差降到小于0.003以下(我的电脑老,且以上参数没有尝试优化过),但在训练100000次左右或更多的时候,识别精度已 经可以达到80%左右,或以上(这和测试样本有关),另外由于创建神经网络时,我让隐层至输出层权值随机生成,就算同样训练100000次,结束时的误差 也不会一样.总之一句,你有足够时间,并且电脑足够快,一是尝试调整学习率和冲量,二是迭代遍数多一点,让误差可以再小一点,这样识别精度可能会更高.          **2.5\*** 为什么上面说是"可能",因为过度追求小的误差会导致过度拟合,这里不展开详述.          **2.6** 训练完后,左边的列表框(Error List)会列出训练过程的一些信息,包括这次训练总共花了多少时间(最上一行),第几次(time)迭代,当前迭代误差(cur\_err),平均误差(avg\_err).  **3.测试BP网络**     测试功能,位于主对话框右上角,我做了两个,上面的一组"select"和"varify"按扭用于测试单张人脸图片,下面一组"select"和"varify"按扭用于批量测试人脸图片,并统计准确率.         **3.1 测试单张图片**           **3.1.1** 点右上第一个"Select"按钮弹出打开文件对话框（图略）:     选中一张没有训练过的人脸图片(pgm格式),点击"Open"按钮(你的电脑可能是"打开"铵钮)            **3.1.2** 然后点击右上的"Varify"按钮,会显示网络对选中图片的测试结果,结果显示在下面的列表框(Varify Result)内,内容如下:  解释一下:  一条横线上面的四行表示: 网络第几个输出单元=值,横线下面的straight表示根据输出层每个单元的值得出的结果:人脸朝向是正面(straight).正确与否可通过与文件名相应信息比较可知.             **3.2 人脸图片批量测试**            **3.2.1** 点右上第二个"Select"按钮弹出打开文件夹对话框(图略):    选择之前存放测试样本集人脸图片的文件夹,如D:\tammo,这个文件夹内都是用于测试的pgm人脸图片. 如果操作无误,按钮左边会显示选中文夹中的第一张人脸图片.             **3.2.2** 然后点击下面的"Varify"按钮,会显示网络对选中文夹内所有人脸图片的测试结果,结果显示在下面的列表框  (Varify Result)内,内容形式如下(图略，受不了了，图全删光了，CSAI的每篇博客容量就30K):      解释一下:由于文夹内可能会有很多人脸文件,于输出信息会有很多,但每个文件的测试输出 信息都如3.1.2节中类似,要说明的有三点:(1)最上面输出被测试文件名.(2)输出结果和正确结果有比较, 如:straight<-->up,左边straight为网络输出结果,右边up为正确结果(从文件名中获取),由于两个内容不同,所以该图片测试结果是错误,即WRONG,如正确则显示RIGHT.(3)最后显示总的正确率,如上图:Accurecy:83.33%.      **4.保存和读取BP网络训练结果** 花了好长时间练训的网络,关闭软件后再打开又要重新再训练,太伤心了,于是我做了保存与读取功能.         **4.1 保存功能**           先要训练好网络,然后点击左下角的"Save"按钮,弹出如下保存对话框（图略）:   选择要保存的径,取个好听的名字,然后点"Save"按钮,搞定.          **4.2 从已保存文件中加载BP网络**           点击左下角的"Load"按钮,弹出如下打开对话框（图略）:   找到相应的.bp后缀的文件,点"Open",加载成功后,下方的"Status:"会显示:Load from 某路径文件名.   **到这里,这个演示程序的使用讲完了,下篇简单讲讲程序代码的实现,包括如何编译源码。** | | | |

5.BP神经网络的实现  
  
BP神经网络的实现嘛,说实话,比较简单,只要你理解了BP的原理,加上一定的编程能力,实现这个算法就不难了,我用CBPNN类封装了BP网络,除了实现基本的BP网络的功能之外,添加了保存和加载网络的功能.目前只实现3层BPNN,下面把源码放上来:  
    5.1 CBPNN类的头文件:  
/\*  
This BPNN class is created by Vincent Gao(c\_gao),2009-10-31.  
you cat use it as any purposes as you want,but you must keep  
this comment on these files,if you have any problems,please   
contact me at:  
  
e-mail:    c\_gao@163.net  
blog:    http://cgao.csai.cn  
  
this class can create a three lays BackPropogation Neural  
Network,to create it you can call the InitBPNN() ,  
to train the network,call Train().  
  
if you want to use this code in your project,please tell me that   
through my e-mail. thanks.  
\*/  
  
#pragma once  
  
＃i nclude <vector>  
＃i nclude <string>  
  
#define DEFAULT\_ETA         0.05        //默认学习率  
#define DEFAULT\_MOMENTUM    0           //默认冲量  
  
  
#define BPNN\_SAVEFILE\_FLAG           "BPNN"  //BP网络保存文件的标记  
#define BPNN\_SAVEFILE\_SEPCHAR        '\n'    //保存文件的分隔符  
  
#define ABS(x)    (x)>0?(x):(-(x))  
  
typedef std::vector<std::pair<std::string,std::vector<double> > >    target\_type;  
  
class CBPNN  
{  
public:  
    CBPNN(void);  
    ~CBPNN(void);  
public:  
    bool InitBPNN(int nInput,int nOutput,int nHidden);  
    bool LoadInputUnit(const double\* data,int count,double scale);  
    bool LoadTarget(const double\* data,int count,double scale=1.0);  
    double Train(double eta,double momentum);  
    void Test(const double\* data,int nDimentions,double scale,std::vector<double>& vecOutput);  
    const double\* GetOutput();  
    bool SaveBPNNFile(const char\* sSavePath,const char\* sTargetName,int nIterateTime,double fStopError,target\_type\* pvecTarget=NULL);  
    bool LoadBPNNFile(const char\* sSavePath,target\_type\* pvecTarget);  
protected:  
    void ReleaseBPNN();  
    double Sigmoid(double x);//激活函数  
    void LayerForward();  
    double OutputError();  
    double HiddenError();  
    void AdjustWeights();  
protected:  
    int m\_nInput;  
    int m\_nOutput;  
    int m\_nHidden;  
  
    double\* m\_InputUnit;  
    double\* m\_HiddenUnit;  
    double\* m\_OutputUnit;  
  
    //即如图(先保存o1单元的所有权,再保存o2的权,上层同理)  
    //       output layer      o  
    //                        / \  
    //       hidden layer   o1  o2  
    //                     /|\  
    //       input layer  o o o  
    //最下层为输入层,最上层为输出层,即网络走向:从下向上  
  
    double\* m\_I2HWeight;  
  
    double\* m\_H2OWeight;  
  
    double\* m\_preI2HWeight;  
    double\* m\_preH2OWeight;  
  
    double\* m\_OutputError;  
    double\* m\_HiddenError;  
  
    double\* m\_Target;  
  
    double m\_eta;//learning rate  
    double m\_momentum;//momentum for updating the weight  
};  
  
简单说明:激活函数使用sigmoid函数,输入层至隐藏层权值保存在m\_I2HWeight内,隐藏层至输出层权值保存在m\_H2OWeight 内,m\_preI2HWeight和m\_preH2OWeight记录上次迭代中的delta权值,用于计算权值更新公式中的冲量 项.m\_OutputError和m\_HiddenError分别保存输出层和隐藏层的误差. m\_InputUnit,m\_HiddenUnit和m\_OutputUnit分别保存输入向量值,隐藏层输出值和输出层输出值,m\_Target用于保 存目标向量值.m\_eta和m\_momentum分别是学习率和冲量.  
  
  
    5.2 CBPNN类的源码(CPP)文件  
  
＃i nclude "stdafx.h" //如果不用VC,或用VC但不用预编译头,请不要include stdafx.h这个文件  
＃i nclude "BPNN.h"  
＃i nclude "time.h"  
＃i nclude "math.h"  
   
   
CBPNN::CBPNN(void)  
{  
       m\_nInput=0;  
       m\_nOutput=0;  
       m\_nHidden=0;  
       m\_InputUnit=NULL;  
       m\_HiddenUnit=NULL;  
       m\_OutputUnit=NULL;  
       m\_I2HWeight=NULL;  
       m\_H2OWeight=NULL;  
       m\_OutputError=NULL;  
       m\_HiddenError=NULL;  
       m\_Target=NULL;  
       m\_preI2HWeight=NULL;  
       m\_preH2OWeight=NULL;  
   
       m\_eta=m\_momentum=0.0;  
}  
   
CBPNN::~CBPNN(void)  
{  
       ReleaseBPNN();  
}  
   
bool CBPNN::InitBPNN(int nInput,int nOutput,int nHidden)  
{  
       ReleaseBPNN();  
   
       m\_nInput=nInput+1;  
       m\_nOutput=nOutput;  
       m\_nHidden=nHidden+1;  
   
       m\_InputUnit = new double[m\_nInput];  
       if(!m\_InputUnit)  
              return false;  
       m\_OutputUnit= new double[m\_nOutput];  
       if(!m\_OutputUnit)  
              return false;  
       m\_HiddenUnit= new double[m\_nHidden];  
       if(!m\_HiddenUnit)  
              return false;  
   
       m\_OutputError= new double[m\_nOutput];  
       if(!m\_OutputError)  
              return false;  
       m\_HiddenError= new double[m\_nHidden];  
       if(!m\_HiddenError)  
              return false;  
   
       m\_Target= new double[m\_nOutput];  
       if(!m\_Target)  
              return false;  
   
   
       memset(m\_InputUnit,0,sizeof(double)\*(m\_nInput));  
       memset(m\_OutputUnit,0,sizeof(double)\*(m\_nOutput));  
       memset(m\_HiddenUnit,0,sizeof(double)\*(m\_nHidden));  
       memset(m\_OutputError,0,sizeof(double)\*(m\_nOutput));  
       memset(m\_HiddenError,0,sizeof(double)\*(m\_nHidden));  
       memset(m\_Target,0,sizeof(double)\*(m\_nOutput));  
   
       int nI2HW=(m\_nHidden-1)\*(m\_nInput);  
       int nH2OW=(m\_nHidden)\*(m\_nOutput);  
   
       m\_I2HWeight= new double[nI2HW];  
       if(!m\_I2HWeight)  
              return false;  
       m\_H2OWeight= new double[nH2OW];  
       if(!m\_H2OWeight)  
              return false;  
   
       srand( (unsigned)time( NULL ) );  
   
       double r=0.0;  
       int i=0;  
       for(i=0; i<nI2HW; i++)  
              m\_I2HWeight[i]=0;//rand()\*1.0/RAND\_MAX;//0.000005;//-0.005+rand()\*1.0/RAND\_MAX;//-0.05到0.05之间  
       for(i=0; i<nH2OW; i++)  
              m\_H2OWeight[i]=rand()\*1.0/RAND\_MAX/100;//rand()\*1.0/RAND\_MAX/10000;//-0.5+rand()\*1.0/RAND\_MAX;  
   
       m\_preI2HWeight= new double[nI2HW];  
       m\_preH2OWeight= new double[nH2OW];  
       memset(m\_preI2HWeight,0,sizeof(double)\*nI2HW);  
       memset(m\_preH2OWeight,0,sizeof(double)\*nH2OW);  
   
//     memset(m\_I2HWeight,0,sizeof(double)\*nInput\*nHidden);  
//     memset(m\_H2OWeight,0,sizeof(double)\*nHidden\*nOutput);  
   
       m\_eta=DEFAULT\_ETA;  
       m\_momentum=DEFAULT\_MOMENTUM;  
   
       return true;  
}  
   
void CBPNN::ReleaseBPNN()  
{  
       if(m\_InputUnit)  
              delete [] m\_InputUnit;  
       if(m\_HiddenUnit)  
              delete [] m\_HiddenUnit;  
       if(m\_OutputUnit)  
              delete [] m\_OutputUnit;  
       if(m\_I2HWeight)  
              delete [] m\_I2HWeight;  
       if(m\_H2OWeight)  
              delete [] m\_H2OWeight;  
       if(m\_OutputError)  
              delete [] m\_OutputError;  
       if(m\_HiddenError)  
              delete [] m\_HiddenError;  
       if(m\_Target)  
              delete[] m\_Target;  
       if(m\_preH2OWeight)  
              delete m\_preH2OWeight;  
       if(m\_preI2HWeight)  
              delete m\_preI2HWeight;  
   
       m\_InputUnit=NULL;  
       m\_HiddenUnit=NULL;  
       m\_OutputUnit=NULL;  
       m\_I2HWeight=NULL;  
       m\_H2OWeight=NULL;  
       m\_OutputError=NULL;  
       m\_HiddenError=NULL;  
       m\_Target=NULL;  
       m\_preH2OWeight=NULL;  
       m\_preI2HWeight=NULL;  
}  
   
bool CBPNN::LoadInputUnit(const double\* data,int count,double scale)  
{  
       if(count != m\_nInput-1)  
              return false;  
   
       //m\_InputUnit[0] 即 x0 始终为 1  
       for(int i=0; i<count; i++)  
              m\_InputUnit[i+1]=data[i]/scale;  
   
       m\_InputUnit[0]=1.0;  
         
       return true;  
}  
   
bool CBPNN::LoadTarget(const double\* data,int count,double scale)  
{  
       if(count != m\_nOutput)  
              return false;  
   
       //m\_Target[0] 即 x0 始终为 1  
       for(int i=0; i<count; i++)  
              m\_Target[i]=data[i]/scale;  
   
       return true;  
}  
   
double CBPNN::Sigmoid(double x)  
{  
       return (1.0 / (1.0 + exp(-x)));  
}  
   
   
void CBPNN::LayerForward()  
{  
       double sum;  
       int i=0,j=0;  
         
       m\_InputUnit[0] = 1.0;  
   
       for (i = 1; i < m\_nHidden; i++)   
       {  
              sum = 0.0;  
              for (j = 0; j < m\_nInput; j++)  
                     sum += m\_I2HWeight[(i-1)\*m\_nInput+j] \* m\_InputUnit[j];//按.h文件中的顺序格式来访问weight[i][j]  
   
              m\_HiddenUnit[i] = Sigmoid(sum);  
       }  
   
       //////////////////////////////////////////////////////////////////////////  
   
       m\_HiddenUnit[0]=1.0;  
       for(i=0; i < m\_nOutput; i++)  
       {  
              sum=0.0;  
              for(j=0; j < m\_nHidden; j++)  
                     sum+=m\_H2OWeight[i\*m\_nHidden+j] \* m\_HiddenUnit[j];  
                
              m\_OutputUnit[i]=Sigmoid(sum);  
       }  
   
}  
   
double CBPNN::OutputError()  
{  
       double o, t, errsum;  
       errsum = 0.0;  
   
       for (int i = 0; i < m\_nOutput; i++)   
       {  
              o = m\_OutputUnit[i];  
              t = m\_Target[i];  
              m\_OutputError[i] = o \* (1.0 - o) \* (t - o);  
              errsum += ABS(m\_OutputError[i]);  
       }  
         
       return errsum;  
}  
   
double CBPNN::HiddenError()  
{  
       double h, sum, errsum;  
   
       errsum = 0.0;  
       //对网络的每个隐藏单元计算它的误差项  
       for (int i = 1; i < m\_nHidden; i++)  
       {  
              h = m\_HiddenUnit[i];  
              sum = 0.0;  
              for (int j= 0; j < m\_nOutput; j++)   
              {  
                     sum += m\_OutputError[j] \* m\_H2OWeight[j\*m\_nOutput+i];//按.h文件中的顺序格式来访问m\_H2OWeight[i][j]  
              }  
              m\_HiddenError[i] = h \* (1.0 - h) \* sum;  
              errsum += ABS(m\_HiddenError[i]);  
       }  
         
       return errsum;  
}  
   
void CBPNN::AdjustWeights()  
{  
       int i=0,j=0;  
       double delta\_w=0.0;  
       for(j=0; j<m\_nOutput; j++)  
       {  
              for(i=0; i<m\_nHidden; i++)  
              {  
                     delta\_w = ( ( m\_eta \* m\_OutputError[j] \* m\_HiddenUnit[i])+(m\_momentum\*m\_preH2OWeight[j\*m\_nHidden+i]) );  
                     m\_H2OWeight[j\*m\_nHidden+i]+=delta\_w;  
                     m\_preH2OWeight[j\*m\_nHidden+i]=delta\_w;  
              }  
       }  
   
       //////////////////////////////////////////////////////////////////////////  
   
       for(j=1; j<m\_nHidden; j++)  
       {  
              for(i=0; i<m\_nInput; i++)  
              {  
                     delta\_w=( ( m\_eta \* m\_HiddenError[j] \* m\_InputUnit[i])+(m\_momentum\*m\_preI2HWeight[(j-1)\*m\_nInput+i]) );  
                     m\_I2HWeight[(j-1)\*m\_nInput+i]+=delta\_w;  
                     m\_preI2HWeight[(j-1)\*m\_nInput+i]=delta\_w;  
              }  
       }  
}  
   
   
double CBPNN::Train(double eta,double momentum)  
{  
       m\_eta=eta;  
       m\_momentum=momentum;  
         
       double err\_o=0.0;  
       double err\_h=0.0;  
       double err=0.0;  
       //1.  
       LayerForward();  
   
       //2.  
       err\_o=OutputError();  
       err\_h=HiddenError();  
   
       //3.  
       AdjustWeights();  
   
       err=err\_o+err\_h;  
   
       return err;  
}  
   
const double\* CBPNN::GetOutput()  
{  
       return m\_OutputUnit;  
}  
   
void CBPNN::Test(const double\* data,int nDimentions,double scale,std::vector<double>& vecOutput)  
{  
       LoadInputUnit(data,nDimentions,scale);  
       LayerForward();  
   
       vecOutput.clear();  
   
       for(int i=0; i<m\_nOutput; i++)  
       {  
              vecOutput.push\_back(m\_OutputUnit[i]);  
       }  
}

接上篇（第二篇），下面是CBPNN类的剩余实现部分，即实现BP神经网络的保存与读取功能。  
  
//将训练好的神经网络保存至文件  
bool CBPNN::SaveBPNNFile(const char\* sSavePath,const char\* sTargetName,int nIterateTime,double fStopError,target\_type\* pvecTarget)  
{  
       FILE\* fp=fopen(sSavePath,"w");  
       if(fp == NULL)  
              return false;  
   
       if(pvecTarget && pvecTarget->size() <=0 )  
              return false;  
       if(pvecTarget &&  ((\*pvecTarget)[0].second).size() <= 0 )  
              return false;  
       if(pvecTarget &&  ((\*pvecTarget)[0].second).size() != m\_nOutput)  
              return false;  
   
       int i=0;  
         
       fprintf(fp,"%s%c",BPNN\_SAVEFILE\_FLAG,BPNN\_SAVEFILE\_SEPCHAR);  
       fprintf(fp,"%s%c",sTargetName,BPNN\_SAVEFILE\_SEPCHAR);//"Direction" or other Target  
       fprintf(fp,"%d%c",nIterateTime,BPNN\_SAVEFILE\_SEPCHAR);//Iterate Time  
       fprintf(fp,"%lf%c",fStopError,BPNN\_SAVEFILE\_SEPCHAR);//fStopError,注意lf符号  
       fprintf(fp,"%lf%c",m\_eta,BPNN\_SAVEFILE\_SEPCHAR);//eta, the learning rate  
       fprintf(fp,"%lf%c",m\_momentum,BPNN\_SAVEFILE\_SEPCHAR);//momentum  
       fprintf(fp,"%d%c",m\_nInput,BPNN\_SAVEFILE\_SEPCHAR);//count of input unit  
       fprintf(fp,"%d%c",m\_nHidden,BPNN\_SAVEFILE\_SEPCHAR);//count of input unit  
       fprintf(fp,"%d%c",m\_nOutput,BPNN\_SAVEFILE\_SEPCHAR);//count of input unit  
         
       int nI2HW=(m\_nHidden-1)\*(m\_nInput);  
       int nH2OW=(m\_nHidden)\*(m\_nOutput);  
   
       fprintf(fp,"%d%c",nI2HW,BPNN\_SAVEFILE\_SEPCHAR);  
       fprintf(fp,"%d%c",nH2OW,BPNN\_SAVEFILE\_SEPCHAR);  
         
       if(pvecTarget != NULL)  
       {  
              for(target\_type::iterator it=pvecTarget->begin(); it!=pvecTarget->end(); it++)  
              {  
                     fprintf(fp,"%s%c",it->first.c\_str(),BPNN\_SAVEFILE\_SEPCHAR);  
                     for(i=0; i<m\_nOutput; i++)  
                            fprintf(fp,"%lf%c",(it->second)[i],BPNN\_SAVEFILE\_SEPCHAR);  
              }  
       }  
         
   
       //保存三层单元值xij  
//    for(i=0; i<m\_nInput; i++)  
//           fprintf(fp,"%lf%c",m\_InputUnit[i],BPNN\_SAVEFILE\_SEPCHAR);  
//    for(i=0; i<m\_nHidden; i++)  
//           fprintf(fp,"%lf%c",m\_HiddenUnit[i],BPNN\_SAVEFILE\_SEPCHAR);  
//    for(i=0; i<m\_nOutput; i++)  
//           fprintf(fp,"%lf%c",m\_OutputUnit[i],BPNN\_SAVEFILE\_SEPCHAR);  
   
       //保存两层权值wij  
       for(i=0; i<nI2HW; i++)  
              fprintf(fp,"%lf%c",m\_I2HWeight[i],BPNN\_SAVEFILE\_SEPCHAR);  
   
       for(i=0; i<nH2OW; i++)  
              fprintf(fp,"%lf%c",m\_H2OWeight[i],BPNN\_SAVEFILE\_SEPCHAR);  
   
   
   
       fclose(fp);  
       return true;  
}  
   
//从文件读取网络的各参数值  
bool CBPNN::LoadBPNNFile(const char\* sSavePath,target\_type\* pvecTarget)  
{  
       FILE\* fp=fopen(sSavePath,"r");  
       if(fp == NULL)  
              return false;  
   
       char temp[100];  
       fscanf(fp,"%s",temp);  
       if(strcmp(temp,BPNN\_SAVEFILE\_FLAG) != 0)  
       {  
              fclose(fp);  
              return false;  
       }  
   
       fscanf(fp,"%s",temp);  
   
       int nIterateTime;  
       double fStopError;  
       double eta;  
       double momentum;  
       int nInput;  
       int nHidden;  
       int nOutput;  
       int nI2HW;  
       int nH2OW;  
       fscanf(fp,"%d",&nIterateTime);  
       fscanf(fp,"%lf",&fStopError);  
       fscanf(fp,"%lf",&eta);  
       fscanf(fp,"%lf",&momentum);  
       fscanf(fp,"%d",&nInput);  
       fscanf(fp,"%d",&nHidden);  
       fscanf(fp,"%d",&nOutput);  
   
       fscanf(fp,"%d",&nI2HW);  
       fscanf(fp,"%d",&nH2OW);  
   
   
       //根据读到的参数创建BP神经网络  
       InitBPNN(nInput-1,nOutput,nHidden-1);  
   
       int i=0;  
   
       std::string target\_name;  
       double ftemp;  
       std::vector<double> target\_vector;  
       if(pvecTarget != NULL)  
       {  
              pvecTarget->clear();  
              for(i=0; i<nOutput; i++)  
              {  
                     fscanf(fp,"%s",temp);  
                     target\_name=temp;  
                     target\_vector.clear();  
                     for(int j=0; j<m\_nOutput; j++)  
                     {  
                            fscanf(fp,"%lf",&ftemp);  
                            target\_vector.push\_back(ftemp);  
                     }  
                     pvecTarget->push\_back(std::pair<std::string,std::vector<double> > (target\_name,target\_vector));  
              }  
       }  
   
   
      for(i=0; i<nI2HW; i++)  
       {  
              fscanf(fp,"%lf",&(m\_I2HWeight[i]));  
//           sprintf(a,"%lf",m\_I2HWeight[i]);  
//           MessageBox(NULL,A2T(a),\_T(""),MB\_OK);  
       }  
       for(i=0; i<nH2OW; i++)  
              fscanf(fp,"%lf%c",&(m\_H2OWeight[i]));  
   
       fclose(fp);  
   
       return true;  
}  
  
简单说明:我注释写得不多,而且一般都是中英文结合(可能包含语法错误^\_^),需要说明的一点是,输入层单元的第一个结点和隐藏层的第一个结点都为偏置单元,注意看 LayerForward()里的两行:m\_InputUnit[0] = 1.0;和m\_HiddenUnit[0] = 1.0; 就是了,偏置的输出永远为1,这里固定就行.而输出层是没有偏置单元的.那么对于上面演示程序启动后,在不改任何参数的情况下(3个隐单元和4个输出单元),构造的网络如下图:



上图中,偏置单元用蓝色实心圆圈表示.其输出固定为1.0  
  
    **5.3 怎么使用该CBPNN类**         很容易,在实例化对象后,分两训练(从文件加载也可以)和测试两步走:  
  
      **5.3.1 训练**                第一步,调用InitBPNN(输入层单元数,输出层单元数,隐层单元数),注意这里的单元数不包括偏置单元.  
                第二步,调用 LoadInputUnit(), 第一参数为指向导入输入层数据的指针,第二个参数为数据的个数(不包括偏置单元),第三个参数是对导入数据的缩放参数,因为,PGM图片是256阶灰度 的,而现在要求(有很多理由)输入层的值为0~1之间的数,于是用255作为第三个参数可将值缩放在0~1之内.  
                第三步,调用 LoadTarget(),参数同LoadInputUnit(),意思也一样.  
                第四步,调用 Train(学习率,冲量参数)  
  
      **5.3.1 测试**               训练网络或从保存的网络加载以后:  
               第一步,调用 Test(), 前三个参数同LoadInputUnit(),最后一个参数是返回型变量,这里用了STL的vector作为输出向量的容器  
               第二步,在第一步调用后,得到返回的第四个参数就是BP网络的测试结果.  
  
写了那么多,也不知道说明白了没有,本来还想讲一下PGM文件图片格式的,现在不想讲了,有机会再讲吧,源代码在压缩包内common文件夹中的PGMFile.cpp和PGMFile.h两个文件内,自己看吧,很简单的.  
  
下面讲一下附件源代码怎么编译使用。  
  
  **6.源代码的编译与运行**压缩包内facerecognization.rar为所有源码,开发环境是VC6,一共两个工程,第一个工程facerecognization是显示 PGM图片的ActiveX控件,第二个工程BPNNTest演示程序,由于我在程序里用了很多STL容器,而VC6对标准STL的支持非常糟糕,因此我 使用是的sgi的stl3.3版,需要下载并配置到VC6内,另外,演示程序BPNNTest工程,是基于WTL的(而不是MFC),因此也要下载WTL并安装.  
  
    **6.1 给VC6配置sgi的stl**  
如果你用的是VC6以后的版本,这步可以跳过,因为VC6以后的版本都对标准STL支持得很高  
sgi-stl是开源且免费的,从sgi官网地地: http://www.sgi.com/tech/stl/download.html 下载下来后,解压至某路径下,然后打开VC6,选菜单Tools->Options,在弹出的对话框内点选Directories标签页,Show directories for:处要选择Include files,然后将刚才stl的路径添加进去,并移到最上面(这样就优先使用sgi的stl,而不是VC6的stl).完成后如下图这样:



    **6.2 安装WTL7.1,请先关掉VC**  
WTL也是开源的,而且还是微软的开源项目,最好下个WTL7.1版: http://www.vckbase.com/tools/downtools.asp?id=131, 如果你用的是VC6以后的版本,可以考虑去sourceforge下载一个最新版的WTL: :http://sourceforge.net/projects/wtl/files/ ,安装过程如下:  
        **6.2.1** 运行下载回来的exe文件或解压zip文件(视你下载的文件类型而定)至某一路径,如D:\  
        **6.2.2** 进入解压后的路径里,找到AppWiz60文件夹,运行里面的setup.js脚本程序,将自动给你的VC添加一个WTL工程模板(新建工程时就会看到).  
        **6.2.3** 退回上一级目录,进入include目录,将其中的所有文件拷入VC6系统include目录内,我的位置在: C:\Program Files\Microsoft Visual Studio\VC98\Include .  
  
至此,搞定,可以编译源码了!