C51实现8051上运行的神经网络

原文作者：Ralph Heymsfeld是[Sully Station Solutions](http://sullystationsolutions.com/)的创始人和负责人。他的兴趣包括人工智能，机器学习，机器人技术和嵌入式系统。他关于其他主题的著作定期出现在这里和整个互联网上。

原文链接：<http://robotics.hobbizine.com/arduinoann.html>

本文译者修改成支持芯圣电子HC0541 系列8051内核的C51代码，并测试通过，修改后的代码在本文结尾，最新的代码在这里下载: <https://github.com/392625227/C51-NN-network>

这个例子程序通过在HC0541上的几分钟的训练， 可以把7段LED的段码显示翻译成相应的数字。88秒训练可以达到90%左右的识别率。代码8KB左右，RAM 1KB。

假设7段LED的采样换成7通道A/D, 就可以解决相对复杂的任务了。

当然训练过程可以在PC上完成，把训练好的网络用51来做识别任务的话，速度还要快几十倍。预估可以用在很多智能传感器等测量、控制领域。

依赖：

1. Keil C + C51支持
2. 芯圣电子HC051 SDK <http://www.holychip.cn/development.php?class_id=102104101>

如果在PC上仿真运行，无需硬件

原文翻译：

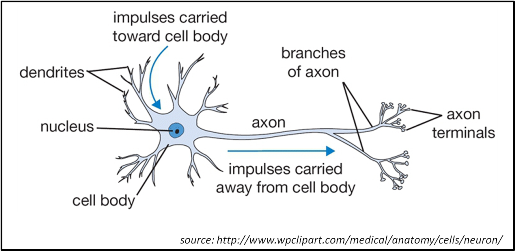
本文介绍了为Arduino Uno微控制器板开发的人工神经网络。这里描述的网络是前馈反向传播网络，可能是最常见的类型。它被认为是用于有监督或无监督学习的良好通用网络。该项目的代码以Arduino草图的形式提供。它是即插即用的-您可以将其上传到Uno并运行它，并且有一部分配置信息可用于快速构建和训练自定义网络。这里提供的文章概述了人工神经网络，草图的详细信息，并介绍了前馈网络和反向传播算法中使用的一些基本概念。

可通过单击以下位置下载该草图：[ArduinoANN.zip](http://robotics.hobbizine.com/ArduinoANN.zip)。

自1980年代中期以来，一直在使用反向传播神经网络。反向传播的基本概念非常简单，尽管算法本身包含一些高阶数学，但不必完全理解如何导出等式以应用它们。在非常小的系统上实现网络存在一些挑战，而在较早的廉价微控制器和业余爱好者主板上，这些挑战是巨大的。但是，Arduinos和今天的许多开发板一样，实际上使这项工作非常短。它的2K SRAM足以用于具有7个输入和4个输出的示例网络，并具有针对多维数组和浮点数学运算的Arduino GCC语言支持，

那有什么用呢？神经网络以身作则。它们已用于从自动驾驶汽车控制到游戏，面部识别到股票市场分析的各种应用中。大多数应用程序将涉及某种类型的模式匹配，在这种情况下，系统的确切输入将是未知的，并且可能缺少或多余的信息。考虑识别手写字符的问题。字母表的一般形状可以提前知道，但是实际输入将始终变化。当然，在ATmega328上构建的小网络并不能完全满足面部识别的任务，但是可以掌握很多在机器人控制和机器学习中进行的实验。

顾名思义，人工神经网络（通常缩写为ANN）是受自然启发的计算模型。这是在某种程度上模仿大脑存储信息并对各种输入做出反应的方式的尝试。在自然界中，神经系统的基本组成部分是一种特殊类型的细胞，称为神经元。

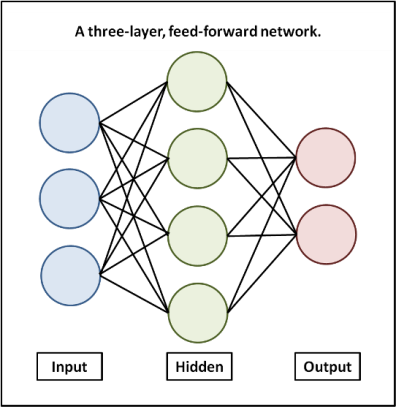


将神经元可视为一个微小的电化学开关可能会很方便，它在受刺激时会打开。神经元在广阔的网络中相互连接。当神经元被刺激激发并变得活跃时，它会沿着该网络发送少量电荷，进而使网络中的其他神经元变得活跃。一个神经元将有多个神经元馈入其中，并且这些连接的强度会有所不同。如果输入有很强的联系，它将提供很多刺激。较弱的连接将提供较少的连接。在非常真实的意义上，神经元可以被认为是将所有这些具有不同强度的输入相加并基于总和产生输出。

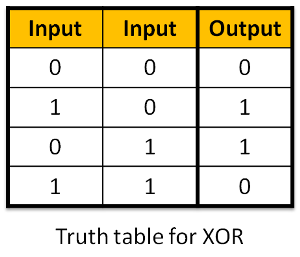
在基于软件的人工神经网络中，神经元及其连接被构造为数学关系。当软件具有输入模式时，它将通过网络馈入该模式，系统地将输入添加到每个神经元，计算该神经元的输出，并使用该输出将适当的输入馈送到网络中的其他神经元。

确定神经元之间的连接强度（也称为权重）成为神经网络应用程序中的首要任务。在反向传播算法中，最初使用随机权重初始化网络。然后为网络提供输入和输出的训练集。当输入通过系统馈送时，会将实际输出与所需输出进行比较，并计算误差。然后，该错误会通过网络反馈，并且权重会根据学习算法进行增量调整。在许多周期（通常是数千个周期）中，最终将训练网络，并在出现输入时给出正确的输出。

在我们在此处构建的前馈网络中，神经元分为三层，分别称为输入层，隐藏层和输出层。一层中的所有神经元都连接到下一层中的所有神经元。下面是这种关系的经典图形表示。



隐藏层在前馈网络中起着至关重要的作用。在早期的神经网络模型中，输入神经元直接连接到输出神经元，并且网络可以实现的解决方案范围极为有限。两层模型无法解决的一个问题是异或逻辑-通常表示为XOR。在布尔逻辑中，当两个输入中的任何一个为true时，XOR关系将为true，但是当两个输入均为true时，则为false。XOR的真值表如下图所示。

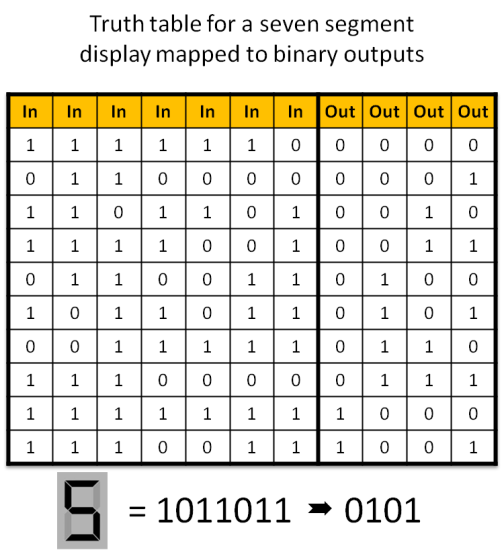


通过在输入和输出之间添加一个层，网络可以解决XOR等问题。一些理论认为，在优化网络的其他条件的情况下，三层网络将能够解决任何真值表。解决XOR是对新网络的良好测试。您会在示例中看到它经常使用，并且它通常被称为神经网络的“ Hello World”程序。

本文随附的草图中实现的网络只是一个演示，实际上并没有执行任何实际功能。草图包括一组训练输入和输出，并且将网络训练到该集合，直到达到预定的准确度为止。此时，草图宣告胜利，然后重新开始。在此过程中，培训结果会定期发送到使用Arduino IDE或任何其他终端程序的串行监视器进行监视的串行端口。（请注意，使用Arduino IDE时，在加载草图后有必要在“工具”菜单中启动串行监视器。）

该程序的结构使得通过简单地在草图开始时更改配置部分中的值就可以非常快速地组装网络和训练集。通过这种设置，可以在不必完全了解所有潜在细微差别的情况下进行网络实验。

配置部分包括两个数据数组Input和Target，它们共同构成训练集的真值表。就其价值而言，草图中设置的训练是一个真值表，该真值表将led数字显示屏（0-9）的七个段映射为二进制数（0000-1001）。您可能会认为这是光学字符识别问题的基本表示。如果研究数组，您会发现它们在输入和输出的映射中提供了丰富的组合，并为网络可以学习解决相当困难的问题提供了很好的概念证明。



要将网络修改为新的训练集，必须在Input和Target数组中输入适当的真值表值，并且还必须在配置部分中调整相应的参数以匹配新的真值表：

* PatternCount：真值表中训练项目或行的数量。
* InputNodes：输入神经元的数量。
* OutputNodes：输出神经元的数量。

您可以在本节中配置一些其他项目进行实验。

* HiddenNodes：隐藏神经元的数量。
* LearningRate：调整实际向后传播该错误的数量。
* 动量：调整前一次迭代的结果对当前迭代的影响。
* InitialWeightMax：设置重量的最大起始值。
* 成功：网络将解决训练集的错误阈值。

作为一个一般概念，HiddenNodes，LearningRate，Momentum和InitialWeightMax一起工作以优化网络以提高学习效率和速度，同时最大程度地减少神经网络设计中遇到的陷阱。

较低的LearningRate值会导致训练过程变慢，但会降低网络陷入振荡的可能性，在这种情况下，网络会持续过度解决训练问题的解决方案，并且永远不会达到成功阈值。在我们的演示中，LearningRate设置为3.。对于大型，非常复杂的网络（比我们在Arduino Uno上构建的网络大得多），该值通常设置得非常低-约为0.01。

动量通过将先前反向传播的一部分注入当前反向传播来平滑训练过程。动量有助于防止网络收敛于一个好的但不是最好的解决方案的现象，也称为收敛于局部最小值。动量值必须在0到1之间。

隐藏神经元的数量会影响网络训练的速度，网络可以解决的问题的复杂性，并有助于防止收敛于局部最小值。您将需要至少具有与输出神经元一样多的隐藏神经元，并且可能需要更多。大量隐藏神经元的缺点是需要存储大量权重。

初始随机权重应该相对较小。草图随附的配置中的InitialWeightMax值为.5。这会将所有初始权重设置为-0.5至.5之间，这是一个很好的起点。

这些参数的理想值根据训练数据的不同而有很大差异，实际上，没有简单的最佳选择方法。经验与反复试验相结合似乎是方法。

配置部分中的最终值“成功”将在系统认为学习训练集时设置系统中错误的阈值级别。这是一个非常小的数字，大于零。这种类型的网络的本质是，系统中的总误差将接近零，但从未真正达到零。

请注意，具有7个输入，8个隐藏的神经元和4个输出的示例网络大约可以在Arduino Uno的2K SRAM上运行。不幸的是，如果您没有在Arduino上用完内存，则不会发出警告，草图的行为只会变得不稳定。好消息是，SRAM分配大于2K的Arduino和Arduino兼容系统的列表一直在增长。如果您成为一名成熟的神经网络实验者，您将有很多选择。

至此，我们已经为您提供了足够的基础，可以将示例网络的代码复制到您自己的计算机上，将其上传到Arduino，并尝试各种设置。

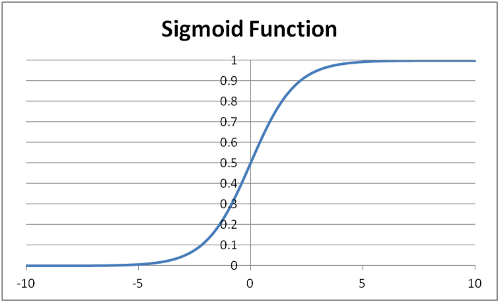
在配置部分之外，我们现在转到草图本身。将神经网络实现为C程序的基本策略是建立数据数组框架，以保持权重并跟踪通过信号前馈和错误后馈累加的累加总数。嵌套的FOR循环序列迭代这些数组，从而在执行反向传播算法时进行所需的各种计算。数组，其他变量和常量的名称与网络中的函数相对应；随着其余说明的进行，这些名称将变得更加清晰。

尽管代码不是绝对的初学者，但是如果您熟悉数组和循环的概念，则希望能够阅读本文随附的草图并遵循逻辑。这是一个高级细分：

* 初始化数组。权重设置为随机数，而两个用于保存反向传播中使用的更改值的其他数组都设置为零。
* 开始一个大循环，通过完整的训练数据集运行系统。
* 随机化训练集在每次迭代中运行的顺序，以减少局部最小值的振荡或收敛。
* 计算隐藏层激活，输出层激活和错误。
* 将错误反向传播到隐藏层。
* 更新权重。
* 如果系统错误大于成功阈值，则运行训练数据的另一次迭代。
* 如果系统错误小于成功阈值，则中断，声明成功，然后将数据发送到串行终端。
* 每1000个周期将训练集的测试运行结果发送到串行终端。

除了编程逻辑外，还有三个要理解的网络基本概念：激活函数，梯度良好和偏置。

激活函数根据馈入该神经元的加权连接的总和来计算该神经元的输出。尽管有变化，但此草图使用了最常见的激活函数，称为“ Sigmoid函数”，因为其独特的形状如下图所示。



该函数的关键特征是无论输入如何，输出都将落在0到1之间。此功能在编码神经网络时非常方便，因为神经元的输出始终可以表示为full on和full之间的范围。关。激活函数可以在代码中采用通用形式的几个地方看到：

|  |
| --- |
| 输出= 1.0 /（1.0 + exp（-Accum）） |

输出是代表已激活神经元输出的数组变量，而Accum是该神经元加权输入的总和。特定公式的复杂性并不重要，除了它们可以方便地产生S型输出。

梯度下降是反向传播的秘密武器。这是一种数学方法，使我们能够计算每个输出神经元的误差幅度，确定与该神经元的每个连接对误差的贡献程度，并对这些连接的权重进行增量调整以系统地减少误差。

梯度下降的第一步是为每个神经元计算一个称为增量的值。增量反映了误差的大小-神经元的目标值与其实际输出之间的差异越大，则增量越大。在输出层，增量计算很简单：

|  |
| --- |
| 增量=（目标-实际）\*实际\*（1-实际） |

在代码中被视为

|  |
| --- |
| OutputDelta [i] =（Target [p] [i]-Output [i]）\* Output [i] \*（1.0-Output [i]）； |

由于没有可测量的目标，因此计算隐藏层的增量将变得更加复杂。取而代之的是，每个隐藏神经元的误差幅度是根据权重与针对输出层计算的增量之间的关系得出的。对于每个隐藏的神经元，代码逐步遍历所有输出连接，将权重乘以增量，并保持运行总计：

|  |
| --- |
| 累计+ = OutputWeights [i] [j] \* OutputDelta [j]； |

然后，通过将存储在Accum中的值替换为在第一次计算中看到的公式中的Target [p] [i]-Output [i]值来计算内层增量：

|  |
| --- |
| HiddenDelta [i] =累计\* Hidden [i] \*（1.0-Hidden [i]）; |

计算出两层的增量值后，下一步就是实际进行操作并调整权重。在这里，我们看到学习率和动量的值如何改变权重的变化。对于每个重量，更改量由以下公式确定：

|  |
| --- |
| 变化=（学习率\*体重\*差异）+（动量\*以前的变化） |

然后通过将旧权重添加到更改值中来找到新权重：

|  |
| --- |
| 体重=体重+变化 |

对于内层和隐藏层之间的权重，公式在代码中显示为：

|  |
| --- |
| ChangeHiddenWeights [j] [i] =学习率\*输入[p] [j] \* HiddenDelta [i] +动量\* ChangeHiddenWeights [j] [i];  HiddenWeights [j] [i] + = ChangeHiddenWeights [j] [i]; |

对于隐藏层和输出层之间的权重，公式在代码中显示为：

|  |
| --- |
| ChangeOutputWeights [j] [i] =学习率\*隐藏[j] \* OutputDelta [i] +动量\* ChangeOutputWeights [j] [i]；  OutputWeights [j] [i] + = ChangeOutputWeights [j] [i]; |

最后，我们开始偏见，这是一个相对简单的概念，但是当不理解它时，可能会使代码有些混乱。输入层和隐藏层每个都包含一个始终触发的额外神经元（换句话说，它始终隐含激活为“ 1”）。偏差值对网络有几个积极影响。它增加了稳定性并扩大了可能的解决方案的数量。最重要的是，它消除了所有输入均为零且因此没有信号通过网络传播的可能性。如果查看包含权重和变化值的数组的声明，则会看到该额外的神经元。还有，你

这样，我们将结束对网络的讨论。如果这里介绍的概念对您来说都是新手，那么毫无疑问，它将需要一些持续的努力才能对反向传播的工作原理形成令人满意的理解。并不缺少与神经网络相关的材料，但是其中很多材料已深入到数学理论中，对初学者尝试掌握基本概念并将其转换为代码没有帮助。

计算机科学伯明翰大学医学院的约翰Bullinaria发布了一个分步指南以实施神经网络用C [www.cs.bham.ac.uk/~jxb/NN/nn.html](http://www.cs.bham.ac.uk/~jxb/NN/nn.html)。本指南是特别有用的资源，我发现在编写本文时它是无价的。

在TEK271.com上可以找到另外两个对入门有用的资源：[http:](http://www.rssc.org/content/introduction-neural-nets-part-1) //www.tek271.com/?about [=](http://www.tek271.com/?about=docs/neuralNet/IntoToNeuralNets.html) docs/neuralNet/IntoToNeuralNets.html 和南加州机器人学会：[http：// www.rssc.org/content/introduction-neural-nets-part-1](http://www.rssc.org/content/introduction-neural-nets-part-1)。

#include <math.h>

#include "HC89F0541.h"

#include <stdio.h>

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Network Configuration - customized per network

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PatternCount 10

#define InputNodes 7

#define HiddenNodes 8

#define OutputNodes 4

//const int PatternCount = Pattern\_Count;

//const int InputNodes = Input\_Nodes;

//const int HiddenNodes = Hidden\_Nodes;

//const int OutputNodes = Output\_Nodes;

const float LearningRate = 0.3;

const float Momentum = 0.9;

const float InitialWeightMax = 0.5;

const float Success = 0.1;//0.0004;

typedef ux8 byte;

const byte Input[PatternCount][InputNodes] = {

{ 1, 1, 1, 1, 1, 1, 0 }, // 0

{ 0, 1, 1, 0, 0, 0, 0 }, // 1

{ 1, 1, 0, 1, 1, 0, 1 }, // 2

{ 1, 1, 1, 1, 0, 0, 1 }, // 3

{ 0, 1, 1, 0, 0, 1, 1 }, // 4

{ 1, 0, 1, 1, 0, 1, 1 }, // 5

{ 0, 0, 1, 1, 1, 1, 1 }, // 6

{ 1, 1, 1, 0, 0, 0, 0 }, // 7

{ 1, 1, 1, 1, 1, 1, 1 }, // 8

{ 1, 1, 1, 0, 0, 1, 1 } // 9

};

const byte Target[PatternCount][OutputNodes] = {

{ 0, 0, 0, 0 },

{ 0, 0, 0, 1 },

{ 0, 0, 1, 0 },

{ 0, 0, 1, 1 },

{ 0, 1, 0, 0 },

{ 0, 1, 0, 1 },

{ 0, 1, 1, 0 },

{ 0, 1, 1, 1 },

{ 1, 0, 0, 0 },

{ 1, 0, 0, 1 }

};

typedef int (\*func1)(int);

typedef int (\*func2)(const char\*, int) reentrant;

typedef int (\*func3)(const char\*, float) reentrant;

typedef struct S {

func1 begin;

func2 println;// = printf;

func2 print;// = printf;

func3 printf;// = printf;

}s\_t;

s\_t Serial;

void Uart\_SendByte(unsigned char buf)

{

SCON &=~ 0x10; //失能UART1接收

SBUF = buf; //发送8位串口数据

while(!(SCON & 0x02));

SCON &=~ 0x02; //清除发送中断标志位

SCON |= 0x10; //UART1接收使能

}

char putchar(char c)

{

Uart\_SendByte(c);

return c;

}

int sendmsg(const char\* str, float d){

printf(str, d);

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* End Network Configuration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

xdata int i, j, p, q, r;

xdata int ReportEvery1000;

xdata int RandomizedIndex[PatternCount];

xdata long TrainingCycle;

xdata float Rando;

xdata float Error;

xdata float Accum;

xdata float Hidden[HiddenNodes];

xdata float Output[OutputNodes];

xdata float HiddenWeights[InputNodes+1][HiddenNodes];

xdata float OutputWeights[HiddenNodes+1][OutputNodes];

xdata float HiddenDelta[HiddenNodes];

xdata float OutputDelta[OutputNodes];

xdata float ChangeHiddenWeights[InputNodes+1][HiddenNodes];

xdata float ChangeOutputWeights[HiddenNodes+1][OutputNodes];

int random(int r){

return rand() % r;

}

void toTerminal()

{

for( p = 0 ; p < PatternCount ; p++ ) {

Serial.println("\n");

Serial.print (" Training Pattern: %d ", p);

//Serial.println (p);

Serial.print (" Input ");

for( i = 0 ; i < InputNodes ; i++ ) {

Serial.print ("%d", Input[p][i]);

Serial.print (" ");

}

Serial.print (" Target ");

for( i = 0 ; i < OutputNodes ; i++ ) {

Serial.print ("%d", Target[p][i]);

Serial.print (" ");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute hidden layer activations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

Accum = HiddenWeights[InputNodes][i] ;

for( j = 0 ; j < InputNodes ; j++ ) {

Accum += Input[p][j] \* HiddenWeights[j][i] ;

}

Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute output layer activations and calculate errors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < OutputNodes ; i++ ) {

Accum = OutputWeights[HiddenNodes][i] ;

for( j = 0 ; j < HiddenNodes ; j++ ) {

Accum += Hidden[j] \* OutputWeights[j][i] ;

}

Output[i] = 1.0/(1.0 + exp(-Accum)) ;

}

Serial.print (" Output ");

for( i = 0 ; i < OutputNodes ; i++ ) {

Serial.printf ("%f", Output[i]);

Serial.print (" ");

}

}

}

void predict(u8 \*input)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute hidden layer activations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

Accum = HiddenWeights[InputNodes][i] ;

for( j = 0 ; j < InputNodes ; j++ ) {

Accum += input[j] \* HiddenWeights[j][i] ;

}

Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute output layer activations and calculate errors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < OutputNodes ; i++ ) {

Accum = OutputWeights[HiddenNodes][i] ;

for( j = 0 ; j < HiddenNodes ; j++ ) {

Accum += Hidden[j] \* OutputWeights[j][i] ;

}

Output[i] = 1.0/(1.0 + exp(-Accum)) ;

Output[i] = Output[i] > 0.8 ? 1:0;

}

Serial.print ("result: ");

for( i = 0 ; i < OutputNodes ; i++ ) {

Serial.printf ("%f", Output[i]);

Serial.print (" ");

}

Serial.print ("\n");

}

void setup(){

Serial.begin = (9600);

Serial.println = sendmsg;

Serial.print = sendmsg;

Serial.printf = sendmsg;

//randomSeed(analogRead(3));

ReportEvery1000 = 1;

for( p = 0 ; p < PatternCount ; p++ ) {

RandomizedIndex[p] = p ;

}

}

void main(){

setup();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initialize HiddenWeights and ChangeHiddenWeights

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

for( j = 0 ; j <= InputNodes ; j++ ) {

ChangeHiddenWeights[j][i] = 0.0 ;

Rando = (float)(random(100))/100;

HiddenWeights[j][i] = 2.0 \* ( Rando - 0.5 ) \* InitialWeightMax ;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Initialize OutputWeights and ChangeOutputWeights

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < OutputNodes ; i ++ ) {

for( j = 0 ; j <= HiddenNodes ; j++ ) {

ChangeOutputWeights[j][i] = 0.0 ;

Rando = (float)(random(100))/100;

OutputWeights[j][i] = 2.0 \* ( Rando - 0.5 ) \* InitialWeightMax ;

}

}

Serial.println("Initial/Untrained Outputs: ");

toTerminal();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Begin training

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( TrainingCycle = 1 ; TrainingCycle < 2147483647 ; TrainingCycle++) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Randomize order of training patterns

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( p = 0 ; p < PatternCount ; p++) {

q = random(PatternCount);

r = RandomizedIndex[p] ;

RandomizedIndex[p] = RandomizedIndex[q] ;

RandomizedIndex[q] = r ;

}

Error = 0.0 ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Cycle through each training pattern in the randomized order

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( q = 0 ; q < PatternCount ; q++ ) {

p = RandomizedIndex[q];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute hidden layer activations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

Accum = HiddenWeights[InputNodes][i] ;

for( j = 0 ; j < InputNodes ; j++ ) {

Accum += Input[p][j] \* HiddenWeights[j][i] ;

}

Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Compute output layer activations and calculate errors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < OutputNodes ; i++ ) {

Accum = OutputWeights[HiddenNodes][i] ;

for( j = 0 ; j < HiddenNodes ; j++ ) {

Accum += Hidden[j] \* OutputWeights[j][i] ;

}

Output[i] = 1.0/(1.0 + exp(-Accum)) ;

OutputDelta[i] = (Target[p][i] - Output[i]) \* Output[i] \* (1.0 - Output[i]) ;

Error += 0.5 \* (Target[p][i] - Output[i]) \* (Target[p][i] - Output[i]) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Backpropagate errors to hidden layer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

Accum = 0.0 ;

for( j = 0 ; j < OutputNodes ; j++ ) {

Accum += OutputWeights[i][j] \* OutputDelta[j] ;

}

HiddenDelta[i] = Accum \* Hidden[i] \* (1.0 - Hidden[i]) ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Update Inner-->Hidden Weights

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < HiddenNodes ; i++ ) {

ChangeHiddenWeights[InputNodes][i] = LearningRate \* HiddenDelta[i] + Momentum \* ChangeHiddenWeights[InputNodes][i] ;

HiddenWeights[InputNodes][i] += ChangeHiddenWeights[InputNodes][i] ;

for( j = 0 ; j < InputNodes ; j++ ) {

ChangeHiddenWeights[j][i] = LearningRate \* Input[p][j] \* HiddenDelta[i] + Momentum \* ChangeHiddenWeights[j][i];

HiddenWeights[j][i] += ChangeHiddenWeights[j][i] ;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Update Hidden-->Output Weights

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for( i = 0 ; i < OutputNodes ; i ++ ) {

ChangeOutputWeights[HiddenNodes][i] = LearningRate \* OutputDelta[i] + Momentum \* ChangeOutputWeights[HiddenNodes][i] ;

OutputWeights[HiddenNodes][i] += ChangeOutputWeights[HiddenNodes][i] ;

for( j = 0 ; j < HiddenNodes ; j++ ) {

ChangeOutputWeights[j][i] = LearningRate \* Hidden[j] \* OutputDelta[i] + Momentum \* ChangeOutputWeights[j][i] ;

OutputWeights[j][i] += ChangeOutputWeights[j][i] ;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Every 1000 cycles send data to terminal for display

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ReportEvery1000 = ReportEvery1000 - 1;

if (ReportEvery1000 == 0)

{

Serial.println("\n");

Serial.println("\n");

Serial.print ("TrainingCycle: %d",TrainingCycle );

//Serial.print (TrainingCycle);

Serial.printf (" Error = %f", Error);

//Serial.println (Error, 5);

toTerminal();

if (TrainingCycle==1)

{

ReportEvery1000 = 999;

}

else

{

ReportEvery1000 = 1000;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* If error rate is less than pre-determined threshold then end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if( Error < Success ) break ;

}

Serial.println("\n");

Serial.println("\n");

Serial.print ("TrainingCycle: %d", TrainingCycle);

//Serial.print (TrainingCycle);

Serial.printf (" Error = %f", Error);

// Serial.println (Error, 5);

toTerminal();

Serial.println ("\n");

Serial.println ("\n");

Serial.println ("Training Set Solved! ");

Serial.println ("--------");

Serial.println ("\n");

Serial.println ("\n");

ReportEvery1000 = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* from a 7 segments input data predict the value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

predict(&Input[3]);

}