# Machine Learning---权值调整公式概率解释

## 引言

在LMS算法的数学解释中，对于里面的权值调整公式的做了数学推导，这次我们将选用逻辑回归，并从概率上解释权值调整公式。

## 选取预测函数

### 1.预测函数

我们使用sigmoid函数作为感知器算法的预测函数，所以我们这里使用了逻辑回归而不是线性回归。

公式1

选用这个函数模型的范围（0，1）。

关键是sigmoid函数的导数为这个给计算带来了很大方便。

### 2.概率

感知器（Perceptron），是神经网络中的一个概念，在1950s由Frank Rosenblatt第一次引入。

### 3.单层感知器

单层感知器（Single Layer Perceptron）是最简单的神经网络。它包含输入层和输出层，而输入层和输出层是直接相连的。

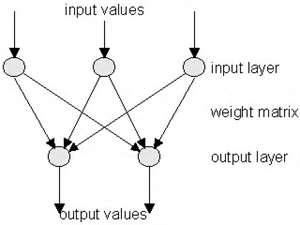


图1.1

图1.1便是一个单层感知器，很简单一个结构，输入层和输出层直接相连。

接下来介绍一下如何计算输出端。

公式（1）

利用公式1计算输出层，这个公式也是很好理解。首先计算输入层中，每一个输入端和其上的权值相乘，然后将这些乘机相加得到乘机和。对于这个乘机和做如下处理，如果乘机和大于临界值（一般是0），输入端就取1；如果小于临界值，就取-1。

以下就给出一段单层感知器的代码。

//////////////////////////////////////////////////////////////////////////

//single layer perceptrons(SLP)

bool slp\_calculate\_output(const double \* inputs,const double \* weights,int nInputs,int & output)

{

if(NULL == inputs || NULL == weights)

return false;

double sum = 0.0;

for (int i = 0 ; i < nInputs ; ++i)

{

sum += (weights[i] \* inputs[i]);

}

//这里我们对乘机和的处理：如果大于0，则输出值为1；其他情况，输出值为-1

if(sum > 0.0)

output = 1;

else

output = -1;

}

//////////////////////////////////////////////////////////////////////////

单层感知器其简单的特性，可以提供快速的计算。它能够实现逻辑计算中的NOT、OR、AND等简单计算。

但是对于稍微复杂的异或就无能无力。下面介绍的多层感知器，就能解决这个问题。

### 4.多层感知器

多层感知器（Multi-Layer Perceptrons），包含多层计算。

相对于单层感知器，输出端从一个变到了多个；输入端和输出端之间也不光只有一层，现在又两层:输出层和隐藏层。

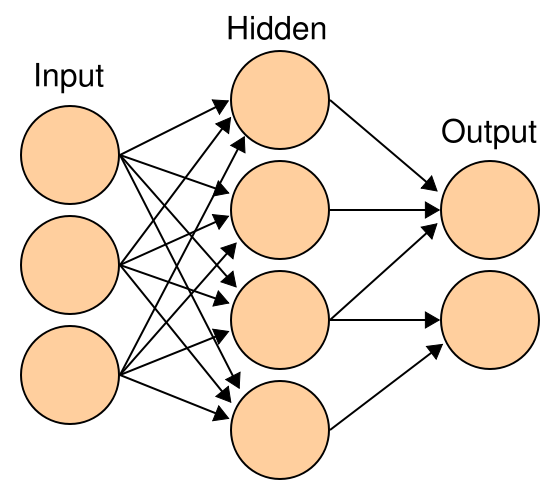


图2.2

图2.2就是一个多层感知器。

对于多层感知器的计算也是比较简单易懂的。首先利用公式1计算每一个。

看一下它代码，就能明白它的工作原理。

//////////////////////////////////////////////////////////////////////////

//Multi-Layer perceptrons(MLP)

const unsigned int nInputs = 4;

const unsigned int nOutputs = 3;

const unsigned int nHiddens = 4;

struct mlp

{

double inputs[nInputs+1];//多一个，存放的bias，一般存放入1

double outputs[nOutputs];

double hiddens[nHiddens+1]; //多一个，存放的bias，一般存放入1

double weight\_hiddens\_2\_inputs[nHiddens+1][nInputs+1];

double weight\_outputs\_2\_hiddens[nOutputs][nHiddens+1];

};

//这里我们对乘机和的处理：如果大于0，则输出值为1；其他情况，输出值为-1

double sigmoid (double val)

{

if(val > 0.0)

return 1.0;

else

return -1.0;

}

//计算输出端

bool mlp\_calculate\_outputs(mlp \* pMlp)

{

if(NULL == pMlp)

return false;

double sum = 0.0;

//首先计算隐藏层中的每一个结点的值

for (int h = 0 ; h < nHiddens ; ++h)

{

double sum = 0.0;

for (int i = 0 ; i < nInputs + 1 ; ++i)

{

sum += pMlp->weight\_hiddens\_2\_inputs[h][i]\*pMlp->inputs[i];

}

pMlp->hiddens[h] = sigmoid (sum);

}

//利用隐藏层作为“输入层”，计算输出层

for (int o = 0 ; o < nOutputs ; ++o)

{

double sum = 0.0;

for (int h = 0 ; h < nHiddens + 1 ; ++h)

{

sum += pMlp->weight\_outputs\_2\_hiddens[o][h]\*pMlp->hiddens[h];

}

pMlp->outputs[o] = sigmoid (sum);

}

return true;

}

//////////////////////////////////////////////////////////////////////////

## 感知器学习算法

### 1.感知器学习

其实感知器学习算法，就是利用第一节介绍的单层感知器。首先利用给的正确数据，计算得到输出值，将输出值和正确的值相比，由此来调整每一个输出端上的权值。

公式2

公式2便是用来调整权值，首先是一个“学习参数”，一般我将它设置成小于1的正数。T便是训练数据中的正确结果，便是第i个输入端的输入值，便是第i个输入端上面的权值。

### 2.代码

对于其介绍，我还是附上代码。

//////////////////////////////////////////////////////////////////////////

//Perceptron Learning Algorithm(PLA)

const unsigned int nTests = 4; //训练数据的数量

const unsigned int nInputs = 2; //输入端的数量

const double alpha = 0.2; //学习参数

struct slp

{

double inputs[nInputs];

double output;

}; //单层感知器

//计算输出值

int compute(double \* inputs,double \* weights)

{

double sum = 0.0;

for (int i = 0 ; i < nInputs; ++i)

{

sum += weights[i]\*inputs[i];

}

//bias

sum += 1.0 \* weights[nInputs];

if(sum > 0.0)

return 1;

else

return -1;

}

//

int \_tmain(int argc, \_TCHAR\* argv[])

{

//正确的训练数据

slp slps[nTests] = {

{-1.0,-1.0,-1.0},

{-1.0, 1.0, 1.0},

{ 1.0,-1.0, 1.0},

{ 1.0, 1.0, 1.0}

};

double weights[nInputs + 1] = {0.0};

bool bLearningOK = false;

//感知器学习算法

while(!bLearningOK)

{

bLearningOK = true;

for (int i = 0 ; i < nTests ; ++i)

{

int output = compute(slps[i].inputs,weights);

if(output != (int)slps[i].output)

{

for (int w = 0 ; w < nInputs ; ++w)

{

weights[w] += alpha \* slps[i].output \* slps[i].inputs[w];

}

weights[nInputs] += alpha \* slps[i].output ;

bLearningOK = false;

}

}

}

for(int w = 0 ; w < nInputs + 1 ; ++w)

{

cout<<"weight"<<w<<":"<<weights[w] <<endl;

}

cout<<"\n";

for (int i = 0 ;i < nTests ; ++i)

{

cout<<"right result：" <<slps[i].output<<"\t";

cout<<"caculate result:" << compute(slps[i].inputs,weights)<<endl;

}

//

char temp ;

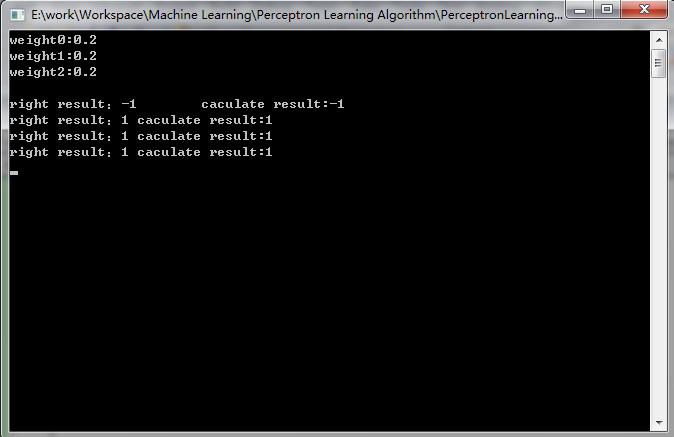
cin>>temp;

return 0;

}

### 2.效果图

下面附上运行效果图



## 总结

感知器学习算法，算是神经网络中的最简单的学习算法。但是通过这个进入学习神经网络学习算法，是个不错的选择。

感知器学习算法，只要是利用了单层感知器。这篇文章中，我们还了解到了另一种感知器：多层感知器。多层感知器主要是用于方向传播学习算法中，这个我后面的文章中会进行介绍。

由于笔者不是专门研究人工智能方面，所以在写这些文章的时候，肯定会有一些错误，也请谅解，上面介绍中有什么错误或者不当地方，敬请指出，不甚欢迎。

如果有兴趣的可以留言，一起交流一下算法学习的心得。

声明：本文章是笔者整理资料所得原创文章，如转载需注明出处，谢谢。