# Machine Learning---Hebbian Learning

## 引言

前面介绍了许多“监督式”学习方式，比如PNN、Backpropagation、LMS等。这些算法有一个共同点：提供的数据中，有目标值。相当于一本带有答案的练习本。接下来将介绍“非监督式”学习方式。

首先就介绍Hebbian Learnig。

## Hebbian Learning 算法基本介绍

### 1.算法思想

这里需要介绍一下Hebb’s Rule。以下来自维基百科：

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

大致意思就是：如果两个神经元，某一个激活了另一个，并且反复持续的激活，那么它们之间的联系值应该被加强。

这背后的意思就是：两个单元有相似的激活值，那么他们的联系将会随着之间的联系而不断加强。

### 2.权值调整公式

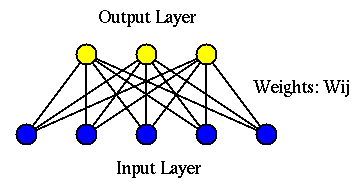
公式（1）

这个公式计算i和j之间的联系权值，是一个学习参数；便是两个神经元的激活值。

如果联系我们之前介绍的神经网络模型，那么我们可以这么建模。

一个非常简单的神经网络结构：一层为输入层，一层为输出端。我们把一个输入端和一个输出端看成上面两个神经元。

大致就如下面这张图所示：



### 3.算法流程

接下来是该算法的流程：

1．将训练数据作为输入端，按照当前的计算模型计算各个输出端的值；

2．将计算得到的输出端的值和输入端的值，按照上面的公式（1），调整各个权值（w）；

3．利用训练数据按照上述方式调整计算模型。

如果新的数据集进入，按照原先的计算模型计算输出值。这个时候，如果新进入的数据集和已分类的某一个patternA相似，那么按照上诉的计算方式调整权值，patternA相对应的权值就会增强，另外的则会减弱。

## 算法实现

下面关键的两个函数

void calculate\_activations()

{

for (int iOut = 0 ; iOut < NUMCELL; ++iOut)

{

double val = 0.0;

for (int iIn = 0 ; iIn < NUMCELL ; ++iIn)

{

val += weight[iIn][iOut] \* input[iIn];

}

if(val > 0.0) output[iOut] = 1;

else

output[iOut] = -1;

}

}

void adjust\_weights()

{

for (int iIn = 0 ; iIn < NUMCELL ; ++iIn)

{

for (int iOut = 0 ; iOut < NUMCELL ; ++iOut)

{

weight[iIn][iOut] += ration \* input[iIn] \* output[iOut];

}

}

}

## 扩充

### 1.Hebbian Learning的局限

比如一个新进入的数据集和当前其中某一个模式patternA完全相同，那么所有的权值都会增加，包括不和patternA对应的权值也会增加，这样就导致计算模型的偏差。

### 2.修改Hebbian Learning

我们将公式1修改为：

公式（2）

下面对这个公式稍加解释：参数和上面公式1中是一样的。这么做之后，可以均衡每一次加强或者减弱权值。

## 总结

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。