

Charlotte77

数学系的数据挖掘民工(公众号:CharlotteDataMining, 深度学习技术交流qq群:339120614)最新深度学习免费学习视频请移步我的B站: <https://www.bilibili.com/video/av75414647>

博客园 首页 新随笔 联系 管理 订阅 

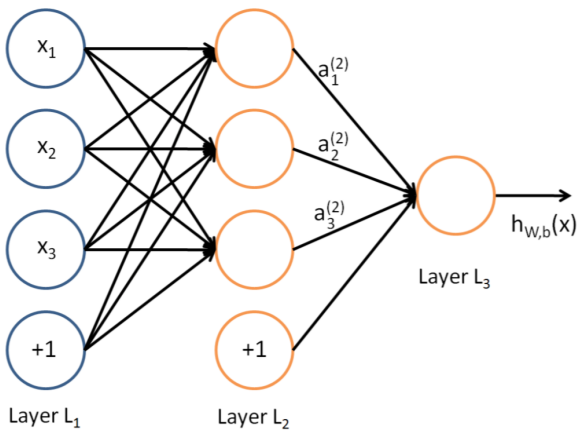
随笔- 57 文章- 0 评论- 1577 阅读- 147万

本博客所有内容以学习、研究和分享为主，如需

一文看懂神经网络中的反向传播法——BackPropagation

最近在看深度学习的东西，一开始看的吴恩达的UFLDL教程，有中文版就直接看了，后来发现有些地方总是不是很明确，又去看英文版，然后又找了些资料看，才发现，中文版的译者在翻译的时候会对省略的公式推导过程进行补充，但是补充的又是错的，难怪觉得有问题。反向传播法其实是神经网络的基础了，但是很多人在学的时候总是会遇到一些问题，或者看到大篇的公式觉得好像很难就退缩了，其实不难，就是一个链式求导法则反复用。如果不想看公式，可以直接把数值带进去，实际的计算一下，体会一下这个过程之后再推导公式，这样就会觉得很容易了。

说到神经网络，大家看到这个图应该不陌生：



这是典型的三层神经网络的基本构成，Layer L1是输入层，Layer L2是隐含层，Layer L3是隐含层，我们现在手里有一堆数据 $\{x_1, x_2, x_3, \dots, x_n\}$ ，输出也是一堆数据 $\{y_1, y_2, y_3, \dots, y_n\}$ ，现在要他们在隐含层做某种变换，让你把数据灌进去后得到你期望的输出。如果你希望你的输出和原始输入一样，那么就是最常见的自编码模型（Auto-Encoder）。可能有人会问，为什么要输入输出都一样呢？有什么用啊？其实应用挺广的，在图像识别，文本分类等等都会用到，我会专门再写一篇Auto-Encoder的文章来说明，包括一些变种之类的。如果你的输出和原始输入不一样，那么就是很常见的人工神经网络了，相当于让原始数据通过一个映射来得到我们想要的输出数据，也就是我们今天要讲的话题。

本文直接举一个例子，带入数值演示反向传播法的过程，公式的推导等到下次写Auto-Encoder的时候再写，其实也很简单，感兴趣的同学可以自己推导下试试：）（注：本文假设你已经懂得基本的神经网络构成，如果完全不懂，可以参考Poll写的笔记：[\[Mechine Learning & Algorithm\] 神经网络基础](#)）

假设，你有这样一个网络层：

转载，请联系本人，标明作者和出处，并且是非商业用途，谢谢！

Email:charlotte77_hu@sina.com
Github:<https://github.com/huxiaoman7>
知乎:https://www.zhihu.com/people/charlotte77_hu
微博:<http://weibo.com/2189505447/profile?topnav=1&wvr=6>
微信公众号:Charlotte数据挖掘



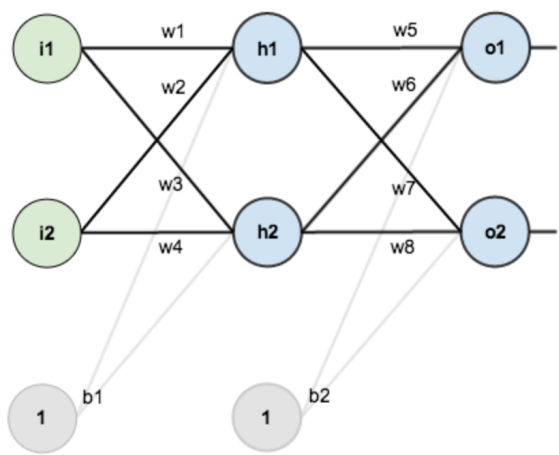
昵称: Charlotte77
园龄: 6年1个月
荣誉: 推荐博客
粉丝: 4098
关注: 8
[+加关注](#)

2022年1月						
<	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

搜索

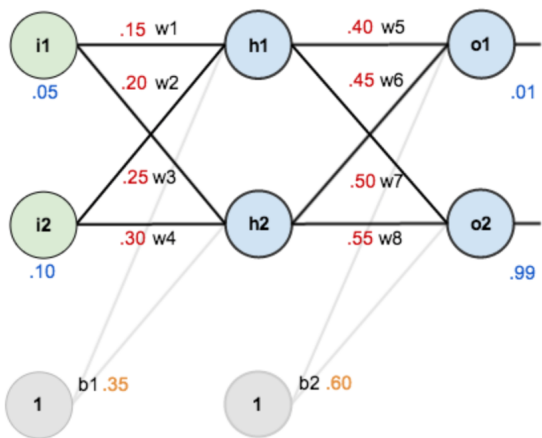
常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)



第一层是输入层，包含两个神经元*i1*，*i2*，和截距项*b1*；第二层是隐含层，包含两个神经元*h1*，*h2*和截距项*b2*，第三层是输出*o1*，*o2*，每条线上标的*w_i*是层与层之间连接的权重，激活函数我们默认为sigmoid函数。

现在对他们赋上初值，如下图：



其中，输入数据 $i_1=0.05$ ， $i_2=0.10$ ；
输出数据 $o_1=0.01$ ， $o_2=0.99$ ；
初始权重 $w_1=0.15$ ， $w_2=0.20$ ， $w_3=0.25$ ， $w_4=0.30$ ；
 $w_5=0.40$ ， $w_6=0.45$ ， $w_7=0.50$ ， $w_8=0.55$

目标：给出输入数据*i1*，*i2*(0.05和0.10)，使输出尽可能与原始输出*o1*，*o2*(0.01和0.99)接近。

Step 1 前向传播

1.输入层---->隐含层：

计算神经元*h1*的输入加权和：

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

神经元*h1*的输出*o1*:(此处用到激活函数为sigmoid函数)：

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

同理，可计算出神经元*h2*的输出*o2*：

$$out_{h2} = 0.596884378$$

2.隐含层---->输出层：

最新随笔

- 1.原创|程序员如何做私活? (一)
- 2.谈谈坚持这件小事
- 3.我在北京这几年 (全)
- 4.【原】深度学习的一些经验总结和建议 | To do v.s Not To Do
- 5.如何高效利用一场技术分享?
- 6.深度学习分布式训练及CTR预估模型应用
- 7.两个月刷完Leetcode前400题经验总结
- 8.【机器学习】如何解决数据不平衡问题
- 9.LeetCode刷题专栏第一篇--思维导图&时间安排
- 10.【资料总结】| Deep Reinforcement Learning 深度强化学习

我的标签

- 深度学习(22)
- 机器学习(10)
- 数据挖掘(5)
- Spark(4)
- 学习心得(3)
- LeetCode(2)
- 年度总结(2)
- 数据挖掘(2)
- 推荐系统(2)
- 文本挖掘(2)
- 更多

积分与排名

积分 - 184646
排名 - 5181

随笔分类 (56)

- Spark(7)
- 机器学习笔记(12)
- 深度学习(23)
- 数据挖掘(9)
- 推荐系统(2)
- 文本挖掘(3)

随笔档案 (57)

- 2020年12月(1)
- 2019年8月(2)
- 2019年7月(2)
- 2019年5月(2)
- 2019年3月(1)
- 2019年2月(1)
- 2019年1月(2)
- 2018年6月(1)
- 2018年5月(1)
- 2018年3月(1)
- 2018年2月(2)
- 2018年1月(4)
- 2017年12月(4)
- 2017年11月(4)
- 2017年10月(2)
- 更多

阅读排行榜

- 1. 一文看懂神经网络中的反向传播法——BackPropagation(360200)
- 2. 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(248379)
- 3. 三个月教你从零入门深度学习(68557)
- 4. 【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络(64076)
- 5. 机器学习基础与实践 (一) ----数据清洗(60413)
- 6. 如何用卷积神经网络CNN识别手写数字集? (38385)

计算输出层神经元o1和o2的值：

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$
$$out_{o2} = 0.772928465$$

这样前向传播的过程就结束了，我们得到输出值为[0.75136079，0.772928465]，与实际值[0.01，0.99]相差还很远，现在我们对误差进行反向传播，更新权值，重新计算输出。

Step 2 反向传播

1.计算总误差

总误差：(square error)

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

但是有两个输出，所以分别计算o1和o2的误差，总误差为两者之和：

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

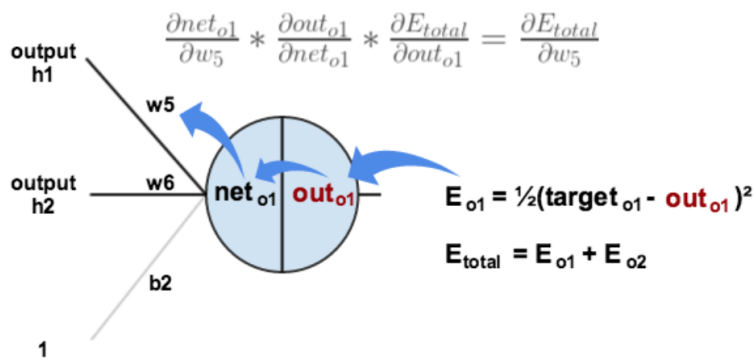
$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

2.隐含层---->输出层的权值更新：

以权重参数w5为例，如果我们想知道w5对整体误差产生了多少影响，可以用整体误差对w5求偏导出：（链式法则）

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

下面的图可以更直观的看清楚误差是怎样反向传播的：



现在我们来分别计算每个式子的值：

计算 $\frac{\partial E_{total}}{\partial out_{o1}}$ ：

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$
$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

7. 机器学习基础与实践（二）----数据转换(33807)
8. 用Tensorflow让神经网络自动创造音乐(29243)
9. 【机器学习】如何解决数据不平衡问题(27822)
10. 【深度学习Deep Learning】资料大全(27703)

评论排行榜

1. 三个月教你从零入门深度学习(222)
2. 我在北京这几年（全）(168)
3. 一文看懂神经网络中的反向传播法——BackPropagation(166)
4. 2015年总结与2016年目标和计划(125)
5. 【深度学习系列】卷积神经网络CNN原理详解（一）——基本原理(115)
6. 2018年总结与2019年目标与计划(90)
7. 【深度学习系列】卷积神经网络详解（二）——自己手写一个卷积神经网络(82)
8. 2017年总结与2018年目标和计划(72)
9. 坑爹的2016年总结(57)
10. 两个月刷完LeetCode前400题经验总结(35)

推荐排行榜

1. 三个月教你从零入门深度学习(243)
2. 一文看懂神经网络中的反向传播法——BackPropagation(189)
3. 【深度学习系列】卷积神经网络CNN原理详解（一）——基本原理(105)
4. 我在北京这几年（全）(104)
5. 2017年总结与2018年目标和计划(52)

最新评论

1. Re:【深度学习系列】卷积神经网络详解（二）——自己手写一个卷积神经网络
博主@夏洛特，你为什么要求输入图片的每个像素值的偏导数？？？我看你一顿操作猛如虎，求了一大波像素值的偏导数，直到我自己写代码实现时候，发现这个偏导好像没什么卵用啊？？。。。不是应该求卷积核的偏导...
--夏天宇
2. Re:我在北京这几年（全）
加油，愿你的事业和财富就像寒武纪生命大爆发一样，多到数不过来
--木子
3. Re:【深度学习系列】卷积神经网络详解（二）——自己手写一个卷积神经网络
交叉熵E求导的部分你没写啊。。。
--夏天宇
4. Re:【深度学习系列】卷积神经网络CNN原理详解（一）——基本原理
“隐含层和最后的输出层的10个神经元连接，就有11760*10=117600个权重w”这里搞错了吧？应该是1510=150吧？？@charlotte77...
--夏天宇
5. Re:【深度学习系列】卷积神经网络CNN原理详解（一）——基本原理
@Charlotte77 “隐含层和最后的输出层的10个神经元连接，就有11760*10=117600个权重w”15个隐藏神经元和10个输出层神经元全连接，怎么看都应该是150个权重参数啊...
--蔡林哲
6. Re:一文看懂神经网络中的反向传播法——BackPropagation
牛
--weilaiwolai
7. Re:【深度学习系列】卷积神经网络详解（二）——自己手写一个卷积神经网络
@Joshua王子 我的理解是这样，既然是取最大值，那就把四个未知数都以加和形式写入，最大项前面系数是1，其余项前面系数是0，所以求导结果为1...
--蔡林哲
8. Re:【深度学习系列】卷积神经网络详解（二）——自己手写一个卷积神经网络

计算 $\frac{\partial out_{o1}}{\partial net_{o1}}$:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

(这一步实际上就是对sigmoid函数求导, 比较简单, 可以自己推导一下)

计算 $\frac{\partial net_{o1}}{\partial w_5}$:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

最后三者相乘:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

这样我们就计算出整体误差E(total)对w5的偏导值。

回过头来再看看上面的公式, 我们发现:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

为了表达方便, 用 δ_{o1} 来表示输出层的误差:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

因此, 整体误差E(total)对w5的偏导公式可以写成:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

如果输出层误差计为负的话, 也可以写成:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

最后我们来更新w5的值:

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

(其中, η 是学习速率, 这里我们取0.5)

同理, 可更新w6,w7,w8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

3.隐含层---->隐含层的权值更新:

方法其实与上面说的差不多, 但是有个地方需要变一下, 在上文计算总误差对w5的偏导时, 是从out(o1)---->net(o1)---->w5,但是在隐含层之间的权值更新时, 是out(h1)---->net(h1)-->w1,而out(h1)会接受E(o1)和E(o2)两个地方传来的误差, 所以这个地方两个都要计算。

想请教卷积层更新参数时候, 权重是filter里的h, 那么偏置b的物理意义是什么呢, 感谢!

--蔡林哲

9. Re:一文看懂神经网络中的反向传播法——BackPropagation

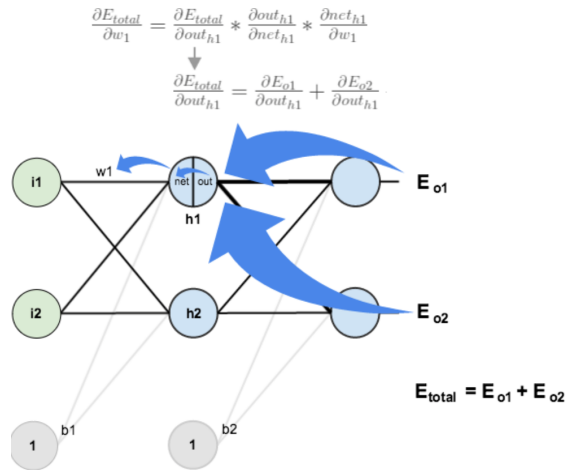
@疯狂de攻城狮 @hothead_hua 这个地方我也有疑问, 更新w1是用更新之前的w5吗 是用更新之前的w5, 因为误差是原始参数产生的, 梯度/导数也是针对原始参数计算的。更新以后的w就是另外一个函...

--yysw28

10. Re:我在北京这几年 (全)

@GScore 哈哈好巧, 我记得你, 当时离职原因确实难以启齿, 对外都说的是我要去读书了 - -...

--Charlotte77



计算 $\frac{\partial E_{total}}{\partial out_{h1}}$:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

先计算 $\frac{\partial E_{o1}}{\partial out_{h1}}$:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

同理, 计算出:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

两者相加得到总值:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

再计算 $\frac{\partial out_{h1}}{\partial net_{h1}}$:

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

再计算 $\frac{\partial net_{h1}}{\partial w_1}$:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

最后, 三者相乘:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

为了简化公式，用 $\sigma(h_1)$ 表示隐含层单元 h_1 的误差：

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}} \right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho} \right) * out_{h1} (1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

最后，更新 w_1 的权值：

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

同理，还可更新 w_2, w_3, w_4 的权值：

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

这样误差反向传播法就完成了，最后我们再把更新的权值重新计算，不停地迭代，在这个例子中第一次迭代之后，总误差 $E(\text{total})$ 由0.298371109下降至0.291027924。迭代10000次后，总误差为0.000035085，输出为[0.015912196, 0.984065734](原输入为[0.01, 0.99])，证明效果还是不错的。

代码(Python):



```
1 #coding:utf-8
2 import random
3 import math
4
5 #
6 # 参数解释:
7 # "pd_" : 偏导的前缀
8 # "d_" : 导数的前缀
9 # "w_ho" : 隐含层到输出层的权重系数索引
10 # "w_ih" : 输入层到隐含层的权重系数的索引
11
12 class NeuralNetwork:
13     LEARNING_RATE = 0.5
14
15     def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights, hidden_layer_bias, output_layer_weights, output_layer_bias):
16         self.num_inputs = num_inputs
17
18         self.hidden_layer = NeuronLayer(num_hidden, hidden_layer_bias)
19         self.output_layer = NeuronLayer(num_outputs, output_layer_bias)
20
21         self.init_weights_from_inputs_to_hidden_layer_neurons(hidden_layer_weights, num_hidden)
22         self.init_weights_from_hidden_layer_neurons_to_output_layer_neurons(output_layer_weights, num_outputs)
23
24     def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights, num_hidden):
25         weight_num = 0
26         for h in range(len(self.hidden_layer.neurons)):
27             for i in range(self.num_inputs):
28                 if not hidden_layer_weights:
29                     self.hidden_layer.neurons[h].weights.append(random.random())
30                 else:
31                     self.hidden_layer.neurons[h].weights.append(hidden_layer_weights[weight_num])
32                 weight_num += 1
33
34     def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self, output_layer_weights, num_outputs):
35         weight_num = 0
36         for o in range(len(self.output_layer.neurons)):
37             for h in range(len(self.hidden_layer.neurons)):
38                 if not output_layer_weights:
39                     self.output_layer.neurons[o].weights.append(random.random())
40                 else:
41                     self.output_layer.neurons[o].weights.append(output_layer_weights[weight_num])
```

```

42         weight_num += 1
43
44     def inspect(self):
45         print('-----')
46         print('* Inputs: {}'.format(self.num_inputs))
47         print('-----')
48         print('Hidden Layer')
49         self.hidden_layer.inspect()
50         print('-----')
51         print('* Output Layer')
52         self.output_layer.inspect()
53         print('-----')
54
55     def feed_forward(self, inputs):
56         hidden_layer_outputs = self.hidden_layer.feed_forward(inputs)
57         return self.output_layer.feed_forward(hidden_layer_outputs)
58
59     def train(self, training_inputs, training_outputs):
60         self.feed_forward(training_inputs)
61
62         # 1. 输出神经元的值
63         pd_errors_wrt_output_neuron_total_net_input = [0] * len(self.output_layer.neurons)
64         for o in range(len(self.output_layer.neurons)):
65
66             #  $\partial E / \partial z_j$ 
67             pd_errors_wrt_output_neuron_total_net_input[o] = self.output_layer.neurons[o].calculate_error(training_outputs[o])
68
69         # 2. 隐含层神经元的值
70         pd_errors_wrt_hidden_neuron_total_net_input = [0] * len(self.hidden_layer.neurons)
71         for h in range(len(self.hidden_layer.neurons)):
72
73             #  $dE/dy_j = \sum \partial E / \partial z_j * \partial z / \partial y_j = \sum \partial E / \partial z_j * w_{ij}$ 
74             d_error_wrt_hidden_neuron_output = 0
75             for o in range(len(self.output_layer.neurons)):
76                 d_error_wrt_hidden_neuron_output += pd_errors_wrt_output_neuron_total_net_input[o] * self.output_layer.neurons[o].weights[h]
77
78             #  $\partial E / \partial z_j = dE/dy_j * \partial z_j / \partial$ 
79             pd_errors_wrt_hidden_neuron_total_net_input[h] = d_error_wrt_hidden_neuron_output
80
81         # 3. 更新输出层权重系数
82         for o in range(len(self.output_layer.neurons)):
83             for w_ho in range(len(self.output_layer.neurons[o].weights)):
84
85                 #  $\partial E_j / \partial w_{ij} = \partial E / \partial z_j * \partial z_j / \partial w_{ij}$ 
86                 pd_error_wrt_weight = pd_errors_wrt_output_neuron_total_net_input[o] * self.output_layer.neurons[o].weights[w_ho]
87
88                 #  $\Delta w = \alpha * \partial E_j / \partial w_{ij}$ 
89                 self.output_layer.neurons[o].weights[w_ho] -= self.LEARNING_RATE * pd_error_wrt_weight
90
91         # 4. 更新隐含层的权重系数
92         for h in range(len(self.hidden_layer.neurons)):
93             for w_ih in range(len(self.hidden_layer.neurons[h].weights)):
94
95                 #  $\partial E_j / \partial w_{ij} = \partial E / \partial z_j * \partial z_j / \partial w_{ij}$ 
96                 pd_error_wrt_weight = pd_errors_wrt_hidden_neuron_total_net_input[h] * self.hidden_layer.neurons[h].weights[w_ih]
97
98                 #  $\Delta w = \alpha * \partial E_j / \partial w_{ij}$ 
99                 self.hidden_layer.neurons[h].weights[w_ih] -= self.LEARNING_RATE * pd_error_wrt_weight
100
101     def calculate_total_error(self, training_sets):
102         total_error = 0
103         for t in range(len(training_sets)):
104             training_inputs, training_outputs = training_sets[t]
105             self.feed_forward(training_inputs)
106             for o in range(len(training_outputs)):
107                 total_error += self.output_layer.neurons[o].calculate_error(training_outputs[o])
108         return total_error
109
110 class NeuronLayer:
111     def __init__(self, num_neurons, bias):
112
113         # 同一层的神经元共享一个截距项b
114         self.bias = bias if bias else random.random()
115
116         self.neurons = []
117         for i in range(num_neurons):
118             self.neurons.append(Neuron(self.bias))

```



```

119
120     def inspect(self):
121         print('Neurons:', len(self.neurons))
122         for n in range(len(self.neurons)):
123             print(' Neuron', n)
124             for w in range(len(self.neurons[n].weights)):
125                 print(' Weight:', self.neurons[n].weights[w])
126             print(' Bias:', self.bias)
127
128     def feed_forward(self, inputs):
129         outputs = []
130         for neuron in self.neurons:
131             outputs.append(neuron.calculate_output(inputs))
132         return outputs
133
134     def get_outputs(self):
135         outputs = []
136         for neuron in self.neurons:
137             outputs.append(neuron.output)
138         return outputs
139
140 class Neuron:
141     def __init__(self, bias):
142         self.bias = bias
143         self.weights = []
144
145     def calculate_output(self, inputs):
146         self.inputs = inputs
147         self.output = self.squash(self.calculate_total_net_input())
148         return self.output
149
150     def calculate_total_net_input(self):
151         total = 0
152         for i in range(len(self.inputs)):
153             total += self.inputs[i] * self.weights[i]
154         return total + self.bias
155
156     # 激活函数sigmoid
157     def squash(self, total_net_input):
158         return 1 / (1 + math.exp(-total_net_input))
159
160
161     def calculate_pd_error_wrt_total_net_input(self, target_output):
162         return self.calculate_pd_error_wrt_output(target_output) * self.calculate_total_net_input_wrt_input()
163
164     # 每一个神经元的误差是由平方差公式计算的
165     def calculate_error(self, target_output):
166         return 0.5 * (target_output - self.output) ** 2
167
168
169     def calculate_pd_error_wrt_output(self, target_output):
170         return -(target_output - self.output)
171
172
173     def calculate_pd_total_net_input_wrt_input(self):
174         return self.output * (1 - self.output)
175
176
177     def calculate_pd_total_net_input_wrt_weight(self, index):
178         return self.inputs[index]
179
180
181 # 文中的例子:
182
183 nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3], hidden_layer_biases=[0.05, 0.1])
184 for i in range(10000):
185     nn.train([0.05, 0.1], [0.01, 0.09])
186     print(i, round(nn.calculate_total_error([[0.05, 0.1], [0.01, 0.09]]), 5))
187
188
189 # 另外一个例子, 可以把上面的例子注释掉再运行一下:
190
191 # training_sets = [
192 #     [[0, 0], [0]],
193 #     [[0, 1], [1]],
194 #     [[1, 0], [1]],
195 #     [[1, 1], [0]]

```



```

196 # ]
197
198 # nn = NeuralNetwork(len(training_sets[0][0]), 5, len(training_sets[0][1]))
199 # for i in range(10000):
200 #     training_inputs, training_outputs = random.choice(training_sets)
201 #     nn.train(training_inputs, training_outputs)
202 #     print(i, nn.calculate_total_error(training_sets))

```

最后写到这里就结束了，现在还不会用latex编辑数学公式，本来都直接想写在草稿纸上然后扫描了传上来，但是觉得太影响阅读体验了。以后会用公式编辑器后再重把公式重新编辑一遍。稳重使用的是sigmoid激活函数，实际还有几种不同的激活函数可以选择，具体的可以参考文献[3]，最后推荐一个在线演示神经网络变化的网址：<http://www.emergentmind.com/neural-network>，可以自己填输入输出，然后观看每一次迭代权值的变化，很好玩~如果有错误的或者不懂的欢迎留言：)

参考文献：

1. Poll的笔记: [\[Machine Learning & Algorithm\] 神经网络基础](#)
(<http://www.cnblogs.com/maybe2030/p/5597716.html#3457159>)
2. Rachel_Zhang: <http://blog.csdn.net/abcjennifer/article/details/7758797>
3. <http://www.cedar.buffalo.edu/%7Esrihari/CSE574/Chap5/Chap5.3-BackProp.pdf>
4. <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

-----本博客所有内容以学习、研究和分享为主，如需转载，请联系本人，标明作者和出处，并且是非商业用途，谢谢！-----

作者: Charlotte77

出处: <http://www.cnblogs.com/charlotte77/>

本文以学习、研究和分享为主，如需转载，请联系本人，标明作者和出处，非商业用途！

关注【Charlotte数据挖掘】回复 '资料' 获取深度学习优质资料

分类: [深度学习](#)

标签: [深度学习](#)



 [Charlotte77](#)
[关注 - 8](#)
[粉丝 - 4098](#)

189 0

推荐博客
[+加关注](#)

« 上一篇: [机器学习基础与实践（二）----数据转换](#)
 » 下一篇: [机器学习基础与实践（三）----数据降维之PCA](#)

posted @ 2016-06-30 16:23 Charlotte77 阅读(360202) 评论(166) 编辑 收藏 举报

< Prev 1 2 3 4

评论

#151楼 2020-05-14 18:31 | ZDL-cnblogs

回复 引用

谢谢