


## 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- 56 文章- 0 评论- 1525

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!

Email:charlotte77\_hu@sina.com

Github:<https://github.com/huxiaoman7>

知乎:[https://www.zhihu.com/people/charlotte77\\_hu](https://www.zhihu.com/people/charlotte77_hu)

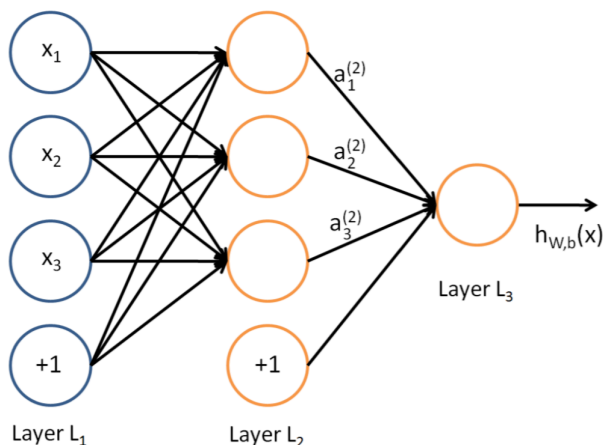
微博:<http://weibo.com/2189505447/profile?topnav=1&wvr=6>

微信公众号:Charlotte数据挖掘

## 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: [\[Machine Learning & Algorithm\] 神经网络基础](#))

假设, 你有这样一个网络层:



昵称: Charlotte77

园龄: 4年8个月

荣誉: 推荐博客

粉丝: 3744

关注: 8

+加关注

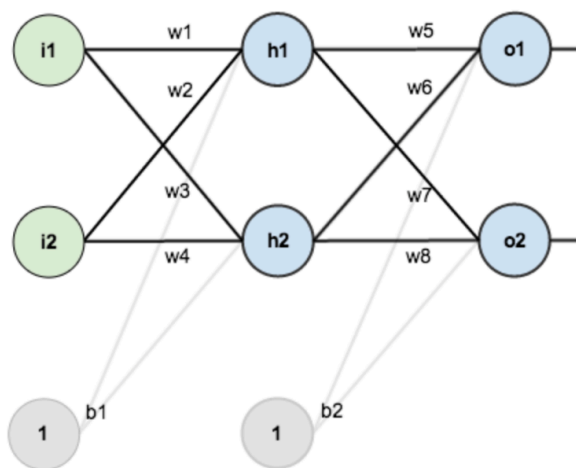
2020年8月						
<	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

## 搜索

<input type="text"/>	找找看
<input type="text"/>	谷歌搜索

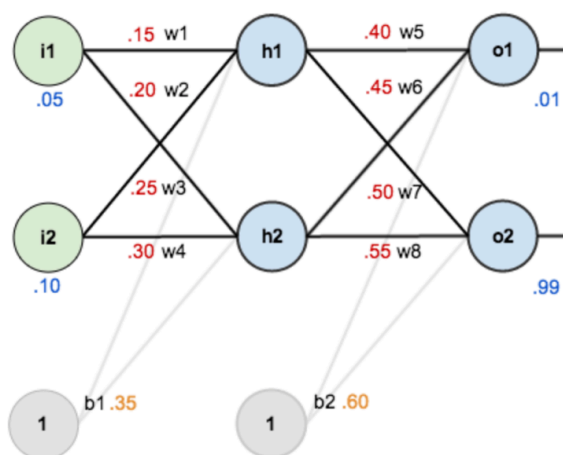
## 常用链接

[我的随笔](#)



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*wi*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  $i1=0.05$ ， $i2=0.10$ ；

输出数据  $o1=0.01$ ， $o2=0.99$ ；

初始权重  $w1=0.15$ ， $w2=0.20$ ， $w3=0.25$ ， $w4=0.30$ ；

$w5=0.40$ ， $w6=0.45$ ， $w7=0.50$ ， $w8=0.55$

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)：

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

我的评论

我的参与

最新评论

我的标签

## 最新随笔

- 1.谈谈坚持这件小事
- 2.我在北京这几年（全）
- 3.【原】深度学习的一些经验总结和建议 | To do v.s Not To Do
- 4.如何高效利用一场技术分享？
- 5.深度学习分布式训练及CTR预估模型应用
- 6.两个月刷完Leetcode前400题经验总结
- 7.【机器学习】如何解决数据不平衡问题
- 8.LeetCode刷题专栏第一篇--思维导图&时间安排
- 9.【资料总结】| Deep Reinforcement Learning 深度强化学习
- 10.2018年总结与2019年目标与计划

## 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)

数据挖掘(2)

推荐系统(2)

文本挖掘(2)

LeetCode(2)

年度总结(2)

更多

## 积分与排名

积分 - 177039

排名 - 3423

## 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 (56)

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)

2017年9月(1)

2016年12月(1)

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：

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求偏导求出：（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：

2016年7月(3)  
2016年6月(3)  
2016年5月(9)  
2016年4月(6)  
2016年3月(1)  
2015年12月(3)

## 最新评论

### 1. Re:谈谈坚持这件小事

感谢，你已经帮助到我了。

--外方

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

隐藏层的误差是等于总误差，还是等于E\_1w5+E\_2w6啊

--Benys

### 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

@风中de石头 谢谢，已经能够运行啦...

--3079779149

### 4. Re:我在北京这几年（全）

很幸运可以看到楼主的文章，关注楼主。大学的第一个暑假，因为疫情待在家好几个月，想在暑假学习深度学习相关知识却因为种种困难犹豫不前，效率低下，希望自己能坚持下去，为了自己的目标，为了成为更好的自己。..

--ywqa

### 5. Re:我在北京这几年（全）

博主的文章很真实，比那些上来就鼓吹理想奋斗的文章的感触更深，能够更理性更客观的认识程序员这个职业，以及在北京一步一步奋斗所经历的点点滴滴。我也要冲向北京了，争取早日秋招上岸！！！！

--MIIEo

### 6. Re:2018年总结与2019年目标与计划

这个复盘好，学习了

--6671

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

Charlotte you are so great Refer to there is back propagation.py that I modified will help you a lot ...

--fatalfeel

### 8. Re:我在北京这几年（全）

姐姐写的很好

--Details\_K

### 9. Re:【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理

您好博主，为啥我感觉一开始最上面那个，参数个数是这样算的：784+(15784+151)+(1015+101)，您 784 1510，乘以10没有看懂是为什么，w^[l]的维数不是(n^[l], n...

--douzujun

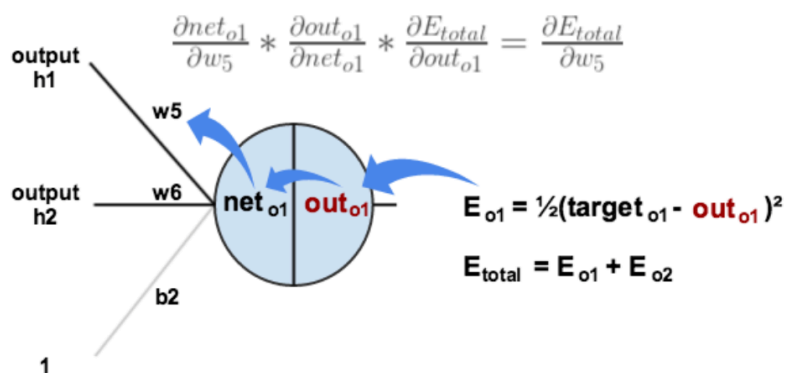
### 10. Re:【深度学习系列】卷积神经网络详解(二)——自己手写一个卷积神经网络

我没看出来有偏置项了呀？

--小艾1

## 阅读排行榜

### 1. 一文看懂神经网络中的反向传播法——BackPropagation(289014)



现在我们来分别计算每个式子的值：

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

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

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

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

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  $\delta_{o1}$  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：

- 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(178792)
- 三个月教你从零入门深度学习(59950)
- 【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(53606)
- 机器学习基础与实践 (一) ----数据清洗(52746)
- 如何用卷积神经网络CNN识别手写数字集？(33469)
- 机器学习基础与实践 (二) ----数据转换(31274)
- 用Tensorflow让神经网络自动创造音乐(27096)
- 【深度学习Deep Learning】资料大全(25457)
- 【原】数据分析/数据挖掘/机器学习---- 必读书目(23378)

## 评论排行榜

- 三个月教你从零入门深度学习(219)
- 我在北京这几年 (全) (161)
- 一文看懂神经网络中的反向传播法——BackPropagation(154)
- 2015年总结与2016年目标和计划(125)
- 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(108)
- 2018年总结与2019年目标与计划(88)
- 2017年总结与2018年目标和计划(72)
- 【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(71)
- 坑爹的2016年总结(57)
- 两个月刷完Leetcode前400题经验总结(33)

## 推荐排行榜

- 三个月教你从零入门深度学习(238)
- 一文看懂神经网络中的反向传播法——BackPropagation(159)
- 我在北京这几年 (全) (97)
- 【深度学习系列】卷积神经网络CNN原理详解(一)——基本原理(96)
- 【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(51)

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

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， $\eta$  是学习速率，这里我们取0.5)

同理，可更新w6,w7,w8:

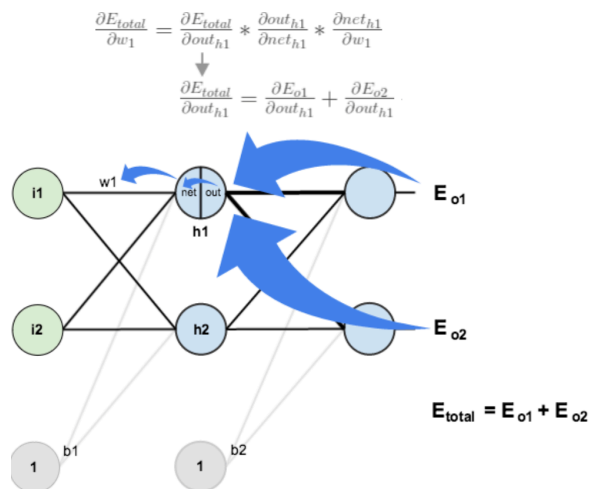
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

### 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。



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(h1)表示隐含层单元h1的误差：

$$\begin{aligned} \frac{\partial E_{total}}{\partial w_1} &= \left( \sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}} \right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1} \\ \frac{\partial E_{total}}{\partial w_1} &= \left( \sum_o \delta_o * w_{ho} \right) * out_{h1}(1 - out_{h1}) * i_1 \\ \frac{\partial E_{total}}{\partial w_1} &= \delta_{h1} i_1 \end{aligned}$$

最后，更新w1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，额可更新w2,w3,w4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重重新计算，不停地迭代，在这个例子中第一次迭代之后，总误差E(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 = None, hidden_layer_bias = None, output_layer_weights = None, output_layer_bias = None):
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)
22         self.init_weights_from_hidden_layer_neurons_to_output_layer_neurons(output_layer_weights)
23
24     def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
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):
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_lay
er.neurons[o].calculate_pd_error_wrt_total_net_input(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_neu
ron_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_hid
den_neuron_output * self.hidden_layer.neurons[h].calculate_pd_total_net_input_wr
t_input()
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_oh 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].calculate_pd_total_net_input_wrt_weight(
w_oh)
87
88                 #  $\Delta w = \alpha * \partial E_j / \partial w_{ij}$ 
89                 self.output_layer.neurons[o].weights[w_oh] -= 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].calculate_pd_total_net_input_wrt_weight(
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_pd_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], hid
den_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55], output_layer_b
ias=0.6)
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]]),
9))
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: [\[Mechine Learning & Algorithm\] 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](#)

好文要顶

关注我

收藏该文

[Charlotte77](#)[关注 - 8](#)[粉丝 - 3744](#)

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](#)

159

0

posted @ 2016-06-30 16:23 [Charlotte77](#) 阅读(289017) 评论(154) 编辑 收藏

&lt; Prev

1

2

3

4

## 评论

#151楼 2020-05-14 18:31 | ZDL-cnblogs

谢谢

支持(0) 反对(0)

#152楼 2020-06-30 19:05 | fatalfeel

Charlotte you are so great

Refer to <https://www.cnblogs.com/charlotte77/p/5629865.html>

there is backpropagation.py that I modified will help you a lot to trace real value

here search and download

<http://www.mediafire.com/file/czit8q113fwv7pt/backpropagation.py>from <https://fatalfeel.blogspot.com/2013/12/ppo-and-awr-guiding.html>

strong suggest you use pycharm to debug python

(the pycharm company is same as android studio)

支持(0) 反对(0)

#153楼 2020-08-03 20:31 | Benys

隐藏层的误差是等于总误差，还是等于 $E_{1w5} + E_{2w6}$ 啊

支持(0) 反对(0)

&lt; Prev

1

2

3

4

[刷新评论](#) [刷新页面](#) [返回顶部](#)注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)， [访问](#) 网站首页。

【推荐】超50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】1200件T恤+6万奖金，阿里云编程大赛报名开启

【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区

【推荐】Java经典面试题整理及答案详解（一）

**相关博文：**

- [第二节，神经网络中反向传播四个基本公式证明——BackPropagation](#)
- [神经网络中的参数的求解：前向和反向传播算法](#)
- [BP算法详解](#)
- [反向传播算法（过程及公式推导）](#)
- [深度学习基础--神经网络--BP反向传播算法](#)
- » [更多推荐...](#)

**最新 IT 新闻：**

- [全国首个蓝牙耳机降噪A级认证！华为FreeLace Pro图赏](#)
- [国产客机新跨越！新舟600首次出口非洲](#)
- [《黑神话：悟空》很酷！但中国3A准备好了吗](#)
- [如何用最少的钱：购买到最划算的流量套餐](#)
- [华为确认：半导体芯片供应受阻！备货3000万手机套片的联发科慌了！](#)
- » [更多新闻...](#)

Copyright © 2020 Charlotte77  
Powered by .NET Core on Kubernetes