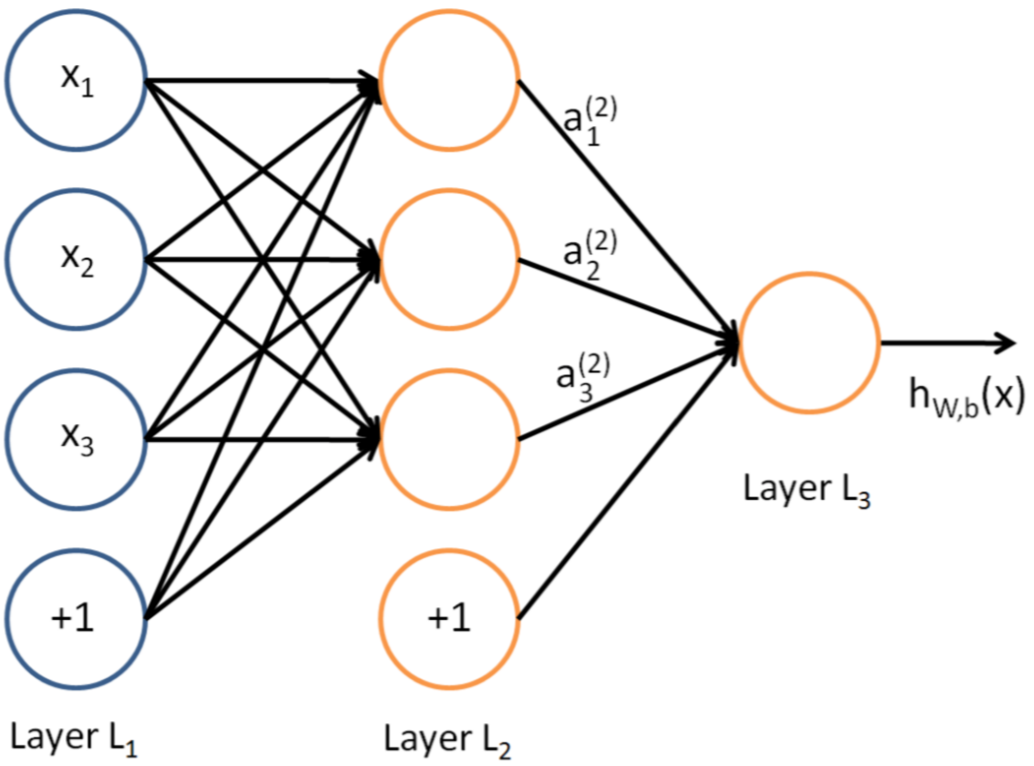
最近在看深度学习的东西，一开始看的吴恩达的UFLDL教程，有中文版就直接看了，后来发现有些地方总是不是很明确，又去看英文版，然后又找了些资料看，才发现，中文版的译者在翻译的时候会对省略的公式推导过程进行补充，但是补充的又是错的，难怪觉得有问题。反向传播法其实是神经网络的基础了，但是很多人在学的时候总是会遇到一些问题，或者看到大篇的公式觉得好像很难就退缩了，其实不难，就是一个链式求导法则反复用。如果不想看公式，可以直接把数值带进去，实际的计算一下，体会一下这个过程之后再来推导公式，这样就会觉得很容易了。

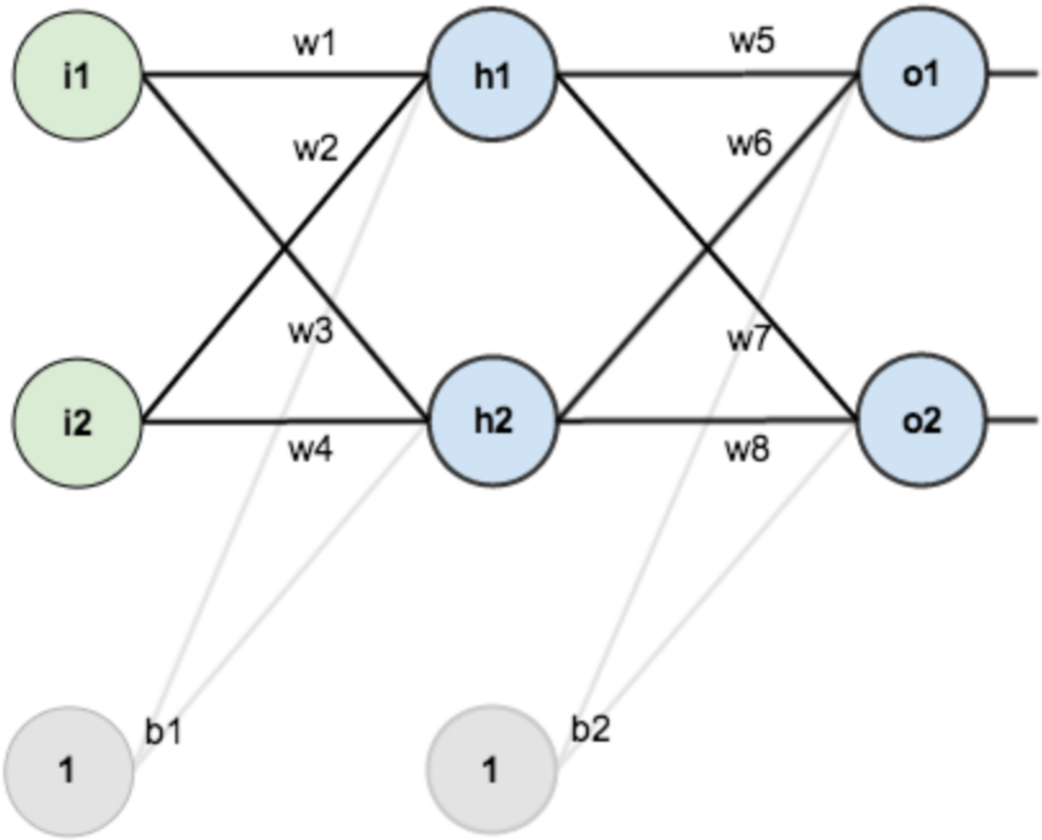
说到神经网络，大家看到这个图应该不陌生：



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{x1,x2,x3,...,xn},输出也是一堆数据{y1,y2,y3,...,yn},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。

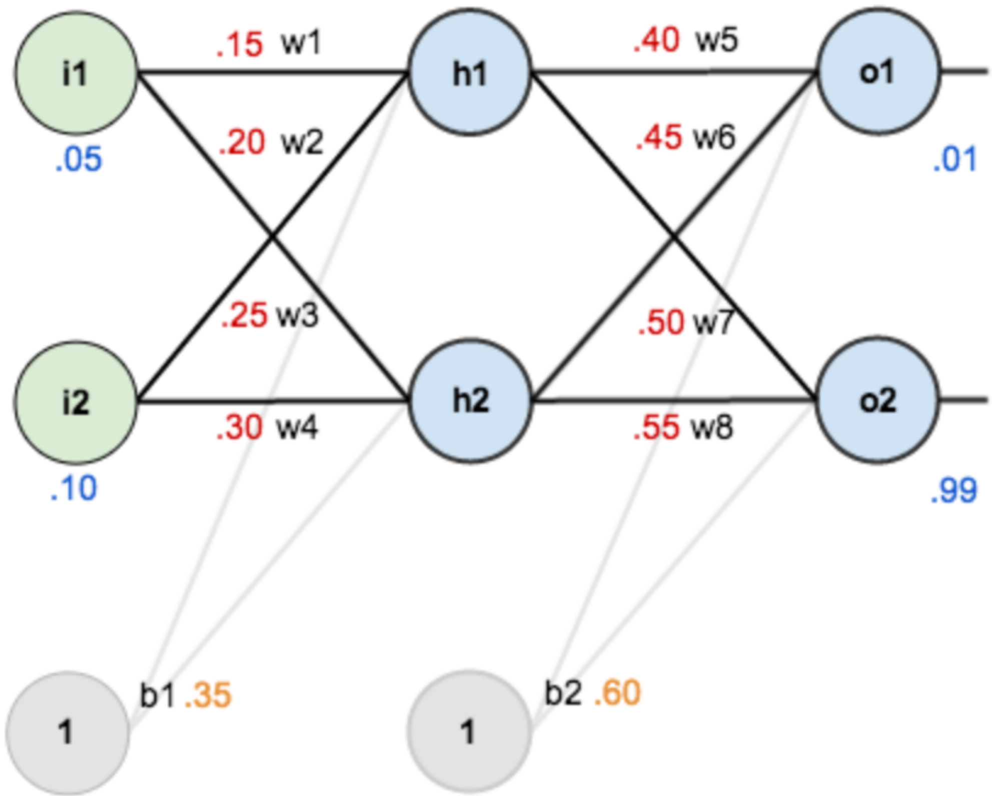
本文直接举一个例子，带入数值演示反向传播法的过程，公式的推导等到下次写Auto-Encoder的时候再写，其实也很简单，感兴趣的同学可以自己推导下试试：）（注：本文假设你已经懂得基本的神经网络构成，如果完全不懂，可以参考Poll写的笔记：[Mechine Learning & Algorithm] 神经网络基础）

假设，你有这样一个网络层：



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)：

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

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：

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

$$E_{o2} = 0.023560026$$

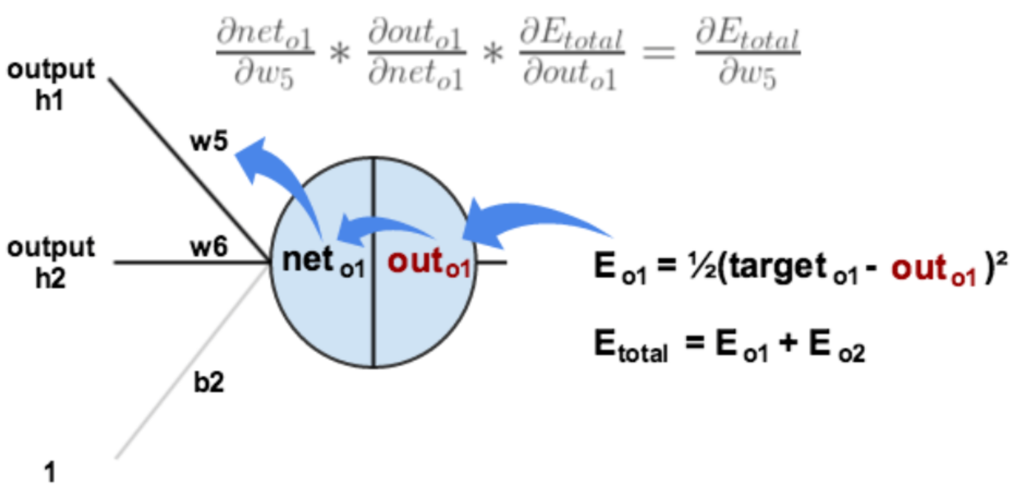
$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

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：



现在我们来分别计算每个式子的值：

计算：

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

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

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

计算：

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

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

（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：

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

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

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，是学习速率，这里我们取0.5）

同理，可更新w6,w7,w8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

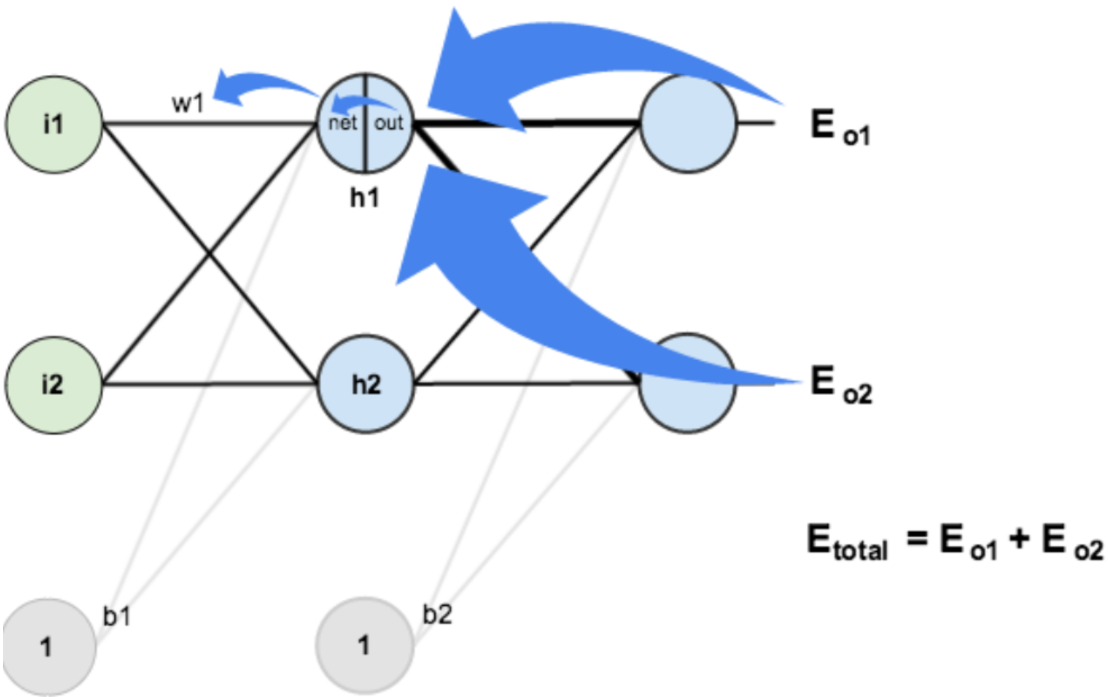
3.隐含层---->隐含层的权值更新：

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。

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

$$\downarrow$$

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



计算：

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

先计算：

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

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

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

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

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

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：

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

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

再计算：

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

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

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：

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

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

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

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(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)：

```
#coding:utf-8
import random
import math

#
#   参数解释：
#   "pd_" ：偏导的前缀
#   "d_" ：导数的前缀
#   "w_ho" ：隐含层到输出层的权重系数索引
#   "w_ih" ：输入层到隐含层的权重系数的索引

class NeuralNetwork:
LEARNING_RATE = 0.5

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):
self.num_inputs = num_inputs

self.hidden_layer = NeuronLayer(num_hidden, hidden_layer_bias)
self.output_layer = NeuronLayer(num_outputs, output_layer_bias)

self.init_weights_from_inputs_to_hidden_layer_neurons(hidden_layer_weights)
self.init_weights_from_hidden_layer_neurons_to_output_layer_neurons(output_layer_weights)

def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
weight_num = 0
for h in range(len(self.hidden_layer.neurons)):
for i in range(self.num_inputs):
ifnot hidden_layer_weights:
self.hidden_layer.neurons[h].weights.append(random.random())
else:
self.hidden_layer.neurons[h].weights.append(hidden_layer_weights[weight_num])
weight_num += 1

def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self, output_layer_weights):
weight_num = 0
for o in range(len(self.output_layer.neurons)):
for h in range(len(self.hidden_layer.neurons)):
ifnot output_layer_weights:
self.output_layer.neurons[o].weights.append(random.random())
else:
```

self.output_layer.neurons[o].weights.append(output_layer_weights[weight_num]) 42 weight_num += 1 43 44def inspect(self): 45print('------') 46print('* Inputs: {}'.format(self.num_inputs)) 47print('------') 48print('Hidden Layer') 49 self.hidden_layer.inspect() 50print('------') 51print('* Output Layer') 52 self.output_layer.inspect() 53print('------') 54 55def feed_forward(self, inputs): 56 hidden_layer_outputs = self.hidden_layer.feed_forward(inputs) 57return self.output_layer.feed_forward(hidden_layer_outputs) 58 59def 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) 64for 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_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) 71for h in range(len(self.hidden_layer.neurons)): 72 73# $dE/dy_j = \Sigma\ \partial E/\partial z_j * \partial z/\partial y_j = \Sigma\ \partial E/\partial z_j * w_{ij}$ 74 d_error_wrt_hidden_neuron_output = 0 75for 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 * self.hidden_layer.neurons[h].calculate_pd_total_net_input_wrt_input() 80 81# 3. 更新输出层权重系数 82for o in range(len(self.output_layer.neurons)): 83for 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].calculate_pd_total_net_input_wrt_weight(w_ho) 87 88# $\Delta w = \alpha * \partial E_j/\partial w_i$ 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 92for h in range(len(self.hidden_layer.neurons)): 93for w_ih in range(len(self.hidden_layer.neurons[h].weights)): 94 95# $\partial E_j/\partial w_i = \partial E/\partial z_j * \partial z_j/\partial w_i$ 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# $\Delta w = \alpha * \partial E_j/\partial w_i$ 99 self.hidden_layer.neurons[h].weights[w_ih] -= self.LEARNING_RATE * pd_error_wrt_weight100101def calculate_total_error(self, training_sets):102 total_error = 0103for t in range(len(training_sets)):104 training_inputs, training_outputs = training_sets[t]105 self.feed_forward(training_inputs)106for o in range(len(training_outputs)):107 total_error += self.output_layer.neurons[o].calculate_error(training_outputs[o])108return total_error109110class NeuronLayer:111def__init__(self, num_neurons, bias):112113# 同一层的神经元共享一个截距项b114 self.bias = bias if bias else random.random()115116 self.neurons = []117for i in range(num_neurons):118 self.neurons.append(Neuron(self.bias))119120def inspect(self):121print('Neurons:', len(self.neurons))122for n in range(len(self.neurons)):123print(' Neuron', n)124for w in range(len(self.neurons[n].weights)):125print(' Weight:', self.neurons[n].weights[w])126print(' Bias:', self.bias)127128def feed_forward(self, inputs):129 outputs = []130for neuron in self.neurons:131 outputs.append(neuron.calculate_output(inputs))132return outputs133134def get_outputs(self):135 outputs = []136for neuron in self.neurons:137 outputs.append(neuron.output)138return outputs139140class Neuron:141def__init__(self, bias):142 self.bias = bias143 self.weights = []144145def calculate_output(self, inputs):146 self.inputs = inputs147 self.output = self.squash(self.calculate_total_net_input())148return self.output149150def calculate_total_net_input(self):151 total = 0152for i in range(len(self.inputs)):153

total += self.inputs[i] * self.weights[i]154return total + self.bias155156# 激活函数 sigmoid157def squash(self, total_net_input):158return 1 / (1 + math.exp(-total_net_input))159160161def calculate_pd_error_wrt_total_net_input(self, target_output):162return self.calculate_pd_error_wrt_output(target_output) * self.calculate_pd_total_net_input_wrt_input();163164# 每一个神经元的误差是由平方差公式计算的165def calculate_error(self, target_output):166return 0.5 * (target_output - self.output) ** 2167168169def calculate_pd_error_wrt_output(self, target_output):170return -(target_output - self.output)171172173def calculate_pd_total_net_input_wrt_input(self):174return self.output * (1 - self.output)175176177def calculate_pd_total_net_input_wrt_weight(self, index):178return self.inputs[index]179180181# 文中的例子:182183 nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3], hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55], output_layer_bias=0.6)184for i in range(10000):185   nn.train([0.05, 0.1], [0.01, 0.09])186print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01, 0.09]]]), 9))187188189#另外一个例子，可以把上面的例子注释掉再运行一下:190191# training_sets = [192#   [[0, 0], [0]],193#   [[0, 1], [1]],194#   [[1, 0], [1]],195#   [[1, 1], [0]]196#]197198# 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/