手写数字识别

by 林伯昱 **时间限制:** 5000 ms **内存限制:** 30000 KB

问题描述

各位都通过了上个月YC公司的面试，加入到了YC公司。入职第一天，诚总看着新员工茂盛的头发及朝气蓬勃的脸庞，甚是满意。诚总拍了拍坐在第一排新人的肩膀，对着大家说：

“各位都是成熟的程序猿了，该学会手写数字识别。这就是炼丹界的hello world，学会它你往后就无往不利了，工资噌噌噌往上涨不是梦想！但做出来太简单了，你们呢要用c/c++来做，用pytorch太方便了没什么难度是不是？当然你们可以参考它源代码的思路，打开函数所在的库文件好好学习，求知若渴，这是我们YC公司的圭臬，知不知道？好了，这就是各位的第一个任务，两周后交出令我满意的答卷！”

在一阵欢送诚总的掌声中，你茫然地转向左边z经理，想问问该怎么办，却发现他已经睡着了。

“啊这...”

“直接卷积网络卷下去吧”另一边传来yd学长的声音说，

“那pooling层怎么处理back propagation？”

“数值逼近啰[旺柴]”

“也可以用MLP吧比较简单，激活函数用sigmoid或tanh就可以直接求导，比较容易实现”身后的xyz说道。

“哇呀呀呀呀好气呀...那个什么我给忘了，给我时间我想想.....”(few years later)“对！就是SGD，随机梯度下降法，随机选取一个样本就更新一次权重训练比较快，毕竟哪知道诚总给的训练集有多大[旺柴]”

“Soga，多谢各位大佬。”

入职典礼结束后，各位都坐到了工位上打算一展身手，但是身为YUC大学优秀毕业生的你，默默掏出了前人留下的版本，已经有隐藏层的结构了。

```
class HiddenLayer{
public:
    HiddenLayer(int in_channel, int out_channel);
    ~HiddenLayer();
    void forward(double* input_data);
    void backward(double* input_data, double* next_layer_delta, double** next_layer_weight, int
next_layer_output);
    void activate_func(double* x);
    double* output_data;
    double* delta;
    int in_channel;
    int out_channel;
    double** weight;
    double* bias;
};
```

/*完整版本请见网络学堂作业附件，请大家在此基础上完成本次实验*/

边看边频频点头，正想着已经赢在了起跑线，却看到部分函数丢失了，留下的... 竟是大大的旺柴。

输入格式

共 2 + N + M 行数据。

第 1 行数据包含训练集样本数 N、测试集样本数 M、标签个数 L，共三个整数。

第 2 行列出 L 个标签值。

接下来 N 行每一行就是 1 个训练样本，依序是 1 个 int 格式标签、784 个浮点数(保留五位小数)。

最后 M 行输入测试样本，每一行共 784 个浮点数。

输出格式

共 M 行，每行一个标签值。

输入样例

```
100 10 2
0 1
0 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.20000 0.62353 0.99216 0.62353 0.19608 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.18824 0.93333 0.98824 0.98824 0.98824 0.92941 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.21176 0.89020 0.99216 0.98824 0.93725 0.91373 0.98824 0.22353 0.02353 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.03922 0.23529
0.87843 0.98824 0.99216 0.98824 0.79216 0.32941 0.98824 0.99216 0.47843 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.63922 0.98824 0.98824 0.98824
0.99216 0.98824 0.98824 0.37647 0.74118 0.99216 0.65490 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.20000 0.93333 0.99216 0.99216 0.74510 0.44706 0.99216
0.89412 0.18431 0.30980 1.00000 0.65882 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.18824 0.93333 0.98824 0.98824 0.70196 0.04706 0.29412 0.47451 0.08235 0.00000
0.00000 0.99216 0.95294 0.19608 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.14902 0.64706 0.99216 0.91373 0.81569 0.32941 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.99216
0.98824 0.64706 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.02745
0.69804 0.98824 0.94118 0.27843 0.07451 0.10980 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.99216 0.98824 0.76471
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.22353 0.98824 0.98824
0.24706 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.99216 0.98824 0.76471 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.77647 0.99216 0.74510 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.99216 0.76863 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.29804 0.96471 0.98824 0.43922 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.99216 0.98824 0.58039 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.33333 0.98824 0.90196 0.09804 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.02745 0.52941 0.99216 0.72941 0.04706 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.33333 0.98824 0.87451 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.02745
0.51373 0.98824 0.88235 0.27843 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
0.00000 0.33333 0.98824 0.56863 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.18824 0.64706 0.98824 0.67843
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.33725
0.99216 0.88235 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.44706 0.93333 0.99216 0.63529 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.33333 0.98824 0.97647
0.57255 0.18824 0.11373 0.33333 0.69804 0.88235 0.99216 0.87451 0.65490 0.21961 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.33333 0.98824 0.98824 0.98824 0.89804
0.84314 0.98824 0.98824 0.98824 0.76863 0.50980 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.10980 0.78039 0.98824 0.98824 0.99216 0.98824 0.98824
0.91373 0.56863 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.09804 0.50196 0.98824 0.99216 0.98824 0.55294 0.14510 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000
/*此处应有100个训练样例及10个测试测试样例，囿于版面所限，仅显示一个样例，完整输入样例见网络学堂。*/
```

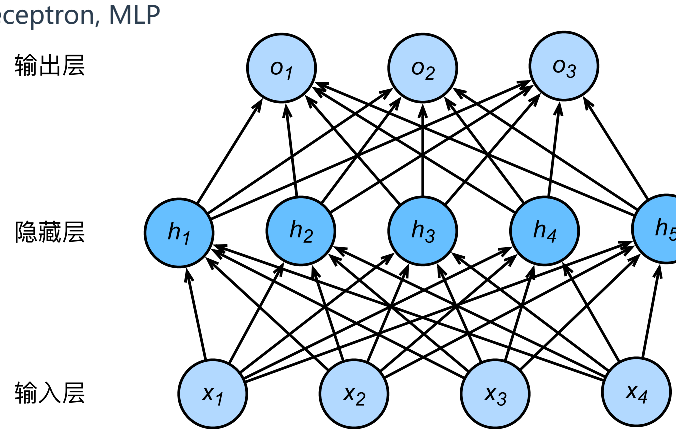
输出样例

```
1
0
1
0
0
1
0
0
1
1
```

提示

40 ≤ N ≤ 6000， M ≤ 500， 2 ≤ L ≤ 6。
训练集中不同标签的样本数量均等；测试集中不同标签的样本数量不一定均等，不会出现不在训练集之中的标签。
每个样本都是 28 * 28 的图片，其值介于 0 - 1。可自行下载MNIST数据集进行测试。
注意到适当选取隐藏层的层数及节点数，过多会导致欠拟合，过少会导致过拟合。
每个测试只要正确率超过90%就视为通过。
前六个测试中，L ≤ 3，后四个测试中 L ≥ 4。

●多层感知机 Multilayer Receptron, MLP



多层感知机在单层神经网络的基础上引入了一到多个隐藏层（hidden layer）。上图展示了含有一个隐藏层的MLP神经网络图，隐藏层位于输入层和输出层之间。
多层感知机中的隐藏层和输出层都是全连接层。由图可见，隐藏层中的神经元和输入层中各个输入完全连接，输出层中的神经元和隐藏层中的各个神经元也完全连接。
设输入维度为n，输出维度为m。全连接层的计算如下：

$$Y_{1\times m} = X_{1\times n} W_{n\times m} + b_{1\times m}$$

以上图为例子，就可视为每条箭头都是一个W权重，h1节点接收到4个输入节点传递的信息，乘上每条边的权重，再加上bias。

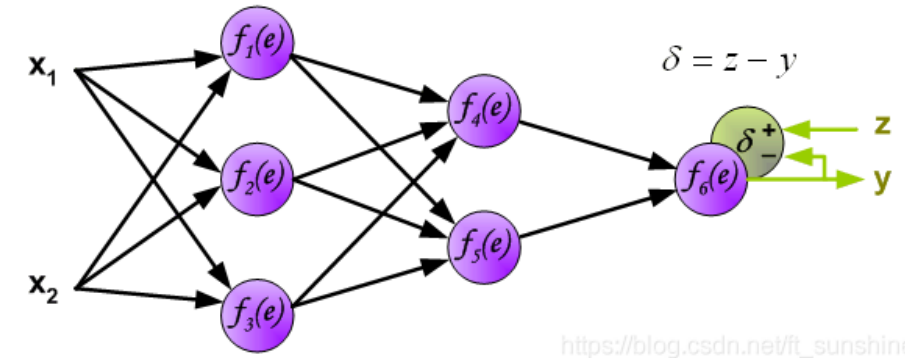
●激活函数 activate function

全连接层只是对数据做仿射变换，而多个仿射变换的叠加仍然是一个仿射变换。解决问题的一个方法是引入非线性变换，例如对隐藏变量使用按元素运算的非线性函数进行变换，然后再作为下一个全连接层的输入。这个非线性函数被称为激活函数。
(此处二段的敘述及图片引用自参考文献，3.8节)

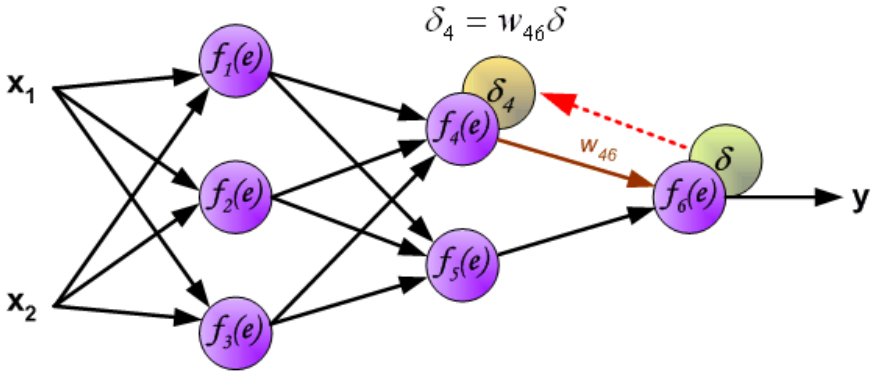
常见的激活函数有ReLU、Sigmoid、tanh、softmax等。

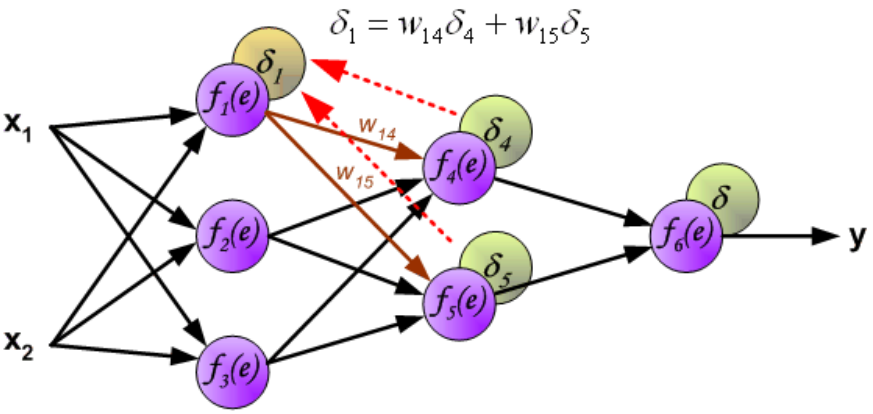
●反向传播 back propagation

一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法。该方法对网络中所有权重计算损失函数的梯度。梯度会反馈给最优化方法，用来更新权值。(此段图片转自参考文献)
y为输出的预测，z为实际label，delta为誤差值。



https://blog.csdn.net/ft_sunshine

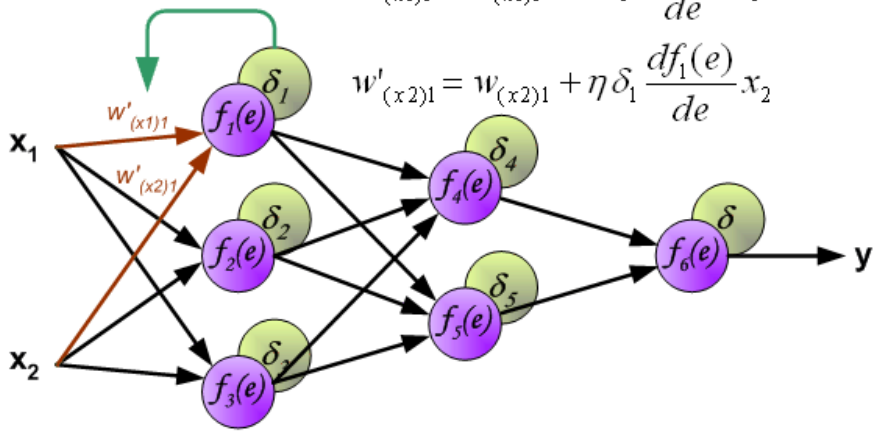




下面开始利用反向传播的误差，计算各个神经元（权重）的导数，开始反向传播修改权重。
w'表示修改後的权重，η是learning rate，δ是誤差，f(x)是激活函数。

$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

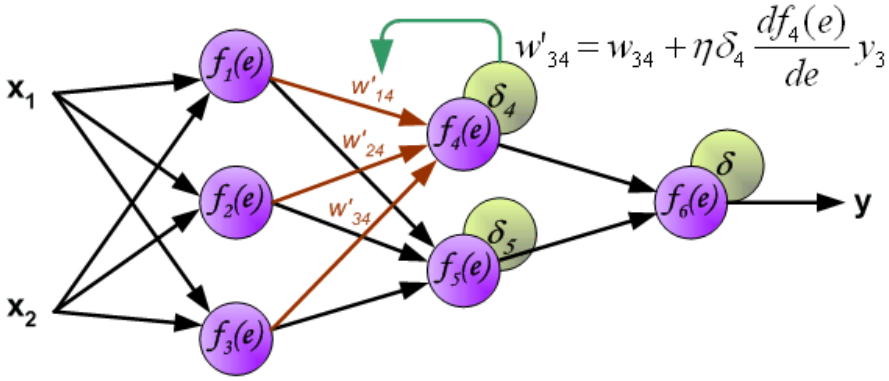
$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

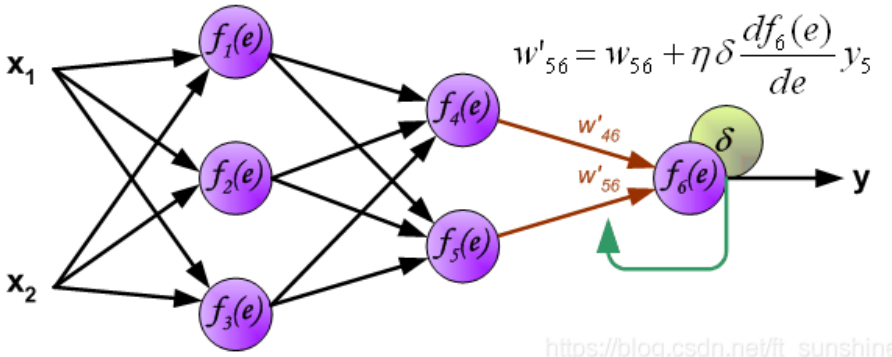
$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



https://blog.csdn.net/ft_sunshine

请输入你的代码:

○ C++03 ○ C++11 ○ C89 ○ C99 ○ C11

Please paste your code here...

提交代码

Copyright © 2017 LambdaHackers. Contact us.