



原创 神经网络实现手写数字识别 (MNIST)

2017-05-10 18:20:42 玄道公子 阅读量 23596 更多

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：<https://blog.csdn.net/xuanwolanxue/article/details/71565934>

一、缘起

原本想沿着 传统递归算法实现迷宫游戏 ——> 遗传算法实现迷宫游戏 ——> 神经网络实现迷宫游戏的思路，在本篇当中也写如何使用神经网络实现迷宫的，但是研究了一下，感觉有些麻烦不太弄，所以就选择了比较常见的方式，实现手写数字识别（所谓的MNIST）。

二、人工神经网络简介

从小至蚂蚁（没有查到具体数目，有的说蚂蚁大脑有25万个神经细胞，也有说是50万个），大至大象，蓝鲸等动物，大脑里面都存在着大量的神经元。人或动物的各种行为或者技能，都或多或少... 是因为这些由数量庞大的神经元连接而成的神经网络组成。而人工神经网络就是仿造生物的大脑来运作的，所以先来了解一笑生物的大脑（神经网络，神经元）。

2.1 生物学神经网络

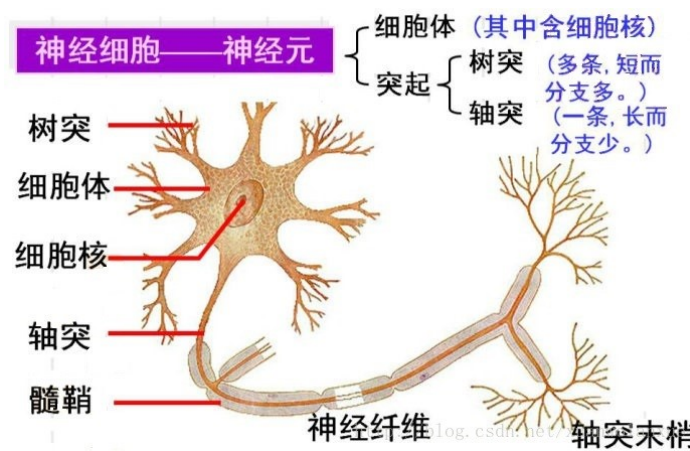
先来看一个表格(来自《游戏编程中的人工智能》一书中第七章第二节，表7.1)：

动物	神经细胞数目 (数量级)
蜗牛	10,000(=10 ⁴)
蜜蜂	100,000 (=10 ⁵)
蜂雀	10,000,000(=10 ⁷)
老鼠	100,000,000(=10 ⁸)
人类	10,000,000,000(=10 ¹⁰)
大象	100,000,000,000(=10 ¹¹)

从上表可以即便小如蜗牛也有数量庞大的神经细胞，自然界的动物需要感知环境，适应环境等等能力，就必须有一套强大的“处理”系统来应对各种突发事件。

从人类和大象的神经细胞数量来看，是否可以推断，表现出来的智能强度，并不一定是和神经细胞的数量成正比的或者说并不是神经细胞越多就越聪明（纯属个人观点）。就好比人工神经网络，机器学习，深度学习等网络一样，并不是层数越多，神经元节点越多越好的。

生物神经细胞如下图所示：



神经细胞之间的信号传递是通过电化学过程完成的，这些信号从一个神经细胞的轴突末梢通过突触（神经细胞轴突末梢与其他神经细胞体，树突相接触的环状或球状结构）传递给下一个神经细胞的树突，然后在从该神经细胞的树突进入细胞体，最后根据实际情况选择是否传递到下一个神经细胞。这里就需要说到神经细胞的两个状态兴奋和不兴奋（即抑制）。神经细胞利用一种我们还不知道的方法把所有从树突进入到细胞体中的信号进行相加，如果信号总和达到某个阈值（注意：是“阈值”，yu，四声，而不是“阀值”哦，以前不知道，问了一下度娘，才知差别很大），就会激发神经细胞进入兴奋状态，这时电信号就会从当前神经细胞，通过其轴突传递向下一个神经细胞，反之则当前神经细胞就会处于抑制状态。

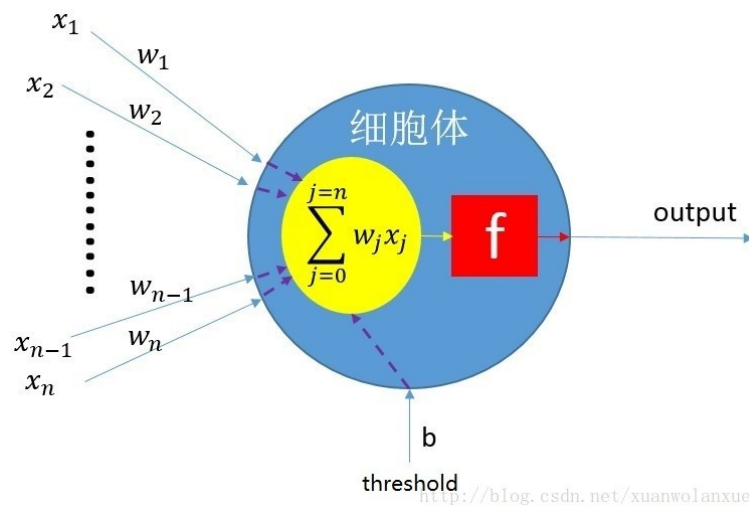
以上只是较为简单或单纯的神经细胞模型，在实际的大脑中电信号在神经细胞传递，不只是从一个细胞的轴突到另一个细胞的树突，还有其他的传递方向：

- 轴突——树突——细胞体
- 轴突——细胞体——树突
- 树突——细胞体——轴突

对于生物学神经网络有兴趣的可以去网上查找相应的资料，这里只是做一个简单介绍，让我们对神经细胞的基本结构以及神经冲动（电信号）传递的基本方式有一个形象的理解。重点还是人工神经网络的说明。

2.2 人工神经网络——神经细胞

人工神经网络 (ANN, Artificial Neuron Network) 是模拟生物大脑的神经网络结构, 它是由许多称为人工神经细胞 (Artificial Neuron, 也称人工神经元) 的细小结构单元组成。这里的人工神经细胞可以看作是生物神经细胞的简化版本或模型。



如上图所示, 就是一个人工神经细胞的示意图, 其中

- $x_1 \dots x_n$: 表示神经细胞的输入, 也就是输入神经细胞的信号。
- $w_1 \dots w_n$: 表示每个输入的权重, 就好比生物神经网络中每个轴突和树突的连接粗细, 强弱的差异。
- b : 偏置权重
- threshold: 偏置 (可以将 $\text{threshold} * b$ 看作是前面提到的生物神经细胞的阈值)
- 蓝色部分: 细胞体。
- 黄色球形是所有输入信号以的求和。
- 红色部分是表示求和之后的信号的激励函数 (即达到阈值就处于兴奋状态, 反之抑制, 当然作为人工神经细胞, 其激励函数很多, 阶跃 (型) 激励函数, sigmoid (s型) 激励函数, 双曲正切 (tanh) 激励函数, ReLu (Rectified Linear Units) 激励函数等等)。

比较原始的感知机(如需要了解什么是感知机, 可问度娘)的数学表达式(或模型)如下:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j < \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j \geq \text{threshold} \end{cases}$$

从上面的表达式可以看出, 在这里, threshold (阈值) 也是神经网络的一个参数。更具体的来说, 要使神经细胞处于兴奋状态 (也就是上面表达式的输出一), 就需要满足如下条件:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n \geq t$$

注意: t : threshold(阈值)

网络在进行学习的过程中, 人工神经网络 (ANN) 的所有权重都需要不断演化 (进化), threshold (阈值) 的数据也不应该例外, 也需要做相应的演化, 所以有必要将threshold (阈值) 转变为权重的形式。从上面的方程两边同时减去 t 得到如下形式:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n - t \geq 0$$

这个方程还可以用另一种形式写出来, 如下:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n + tb \geq 0$$

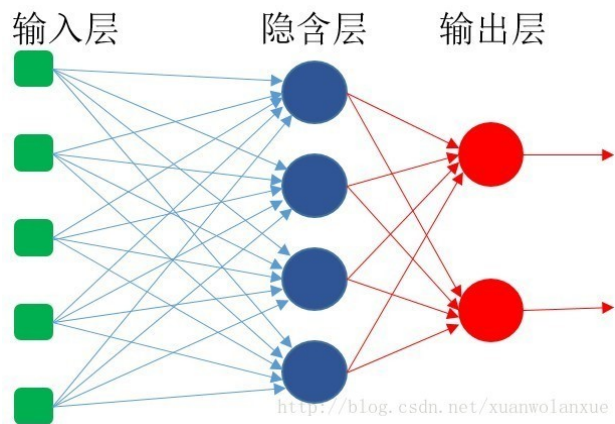
注意: 其中 b 可以取 1 或 -1, 取 1 时, threshold (阈值 t) 取负值就可以了。

也就是说, 可以将 b 看作一个像 w 的权重, 将 t 看作像 x 的输入, 其方程可以进一步变为:

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n + w_b x_b \geq 0$$

2.3 人工神经网络——神经细胞层

动物大脑里的生物神经细胞和其他神经细胞是相互连接在一起的, 从而形成了庞大而复杂的神经网络。同理要创建人工神经网络 (ANN), 人工神经细胞也需要像生物神经细胞那样连接在一起。个神经细胞可以看成是一个点, 大量的神经细胞, 就意味着大量的点, 要将这些点全部连接起来, 可以有很多种连接形式, 其中最容易理解并且也是应用也最广泛的是: 把神经细胞一层一层的连在一起, 如下图所示:



这种类型的神经网络被称为：前馈网络（feedforward network）。这是因为网络的每一层神经细胞的输出都向前馈送（feed）到他们的下一层（如上图中的从左至右的顺序），直至获得整个网络的最终输出。

由上图可知，该网络一共有3层，输入层的每个输入都馈送到了隐含层，作为该层每一个神经细胞的输入；然后从隐含层的每个神经细胞的输出又再次馈送到它的下一层（即输出层）。

上图中只画了一个隐含层，作为前馈网络，理论上可以有任意多个隐含层，但在处理大多数问题时，通常一个隐含就足够了，而有一些更简单的问题，甚至连隐含层都不需要，直接就是一个输入层和输出层。

注意：这里只是说的简单的全连接神经网络（也就是当前层的每一个神经细胞都的输入都包含前一层的所有神经细胞的输出），对于卷积网络（CNN），递归网络（RNN），生成对抗网络（GANs），深度学习，强化学习等，不在讨论的范围之内。

2.4 人工神经网络——演化

一个神经网络在创建之后，要能正常的工作，还是要进行一些演化（或者进化，或者学习），就像动物或者人进行学习或练习一样。对于神经网络来说，演化的过程就是不断的调整每一层，每一个神经细胞的输入权重。能够调节神经网络所有神经细胞的输入权重的方式有很多，这里只说**反向传播（bp: back propagation）**法，下面就详细的说说什么是反向传播法：

首先，我们要来了解一下，为什么网络需要演化，需要学习。答案当然就是：人工神经网络的输出结果与我们的预期不符，存在偏差。

其次，如何演化，如2.3节中的三层神经网络可知，一个神经网络可能会有很多层，每一层有很多神经细胞，每一个神经细胞都有很多输入，与这些输入对应的就是很多权重。如何才能每一个权重都被调整到呢，反向传播算法是这样做的，首先得到网络的输出层的输出与我们期望之间的误差，然后再将这个误差值反向传播到前一层（隐含层），如此往复，最后就是根据误差，学习率以及当前取值点的梯度来更新每层神经细胞的输入权重，这就是所谓的**反向传播算法**。而不断调整神经细胞的权重以便输出误差达到最小的过程叫做**梯度下降**（这是一个寻找极小值的过程，也就是寻找一个偏导数斜率最小的点）。

在百度中查了一下梯度的解释：在单变量的实值函数的情况，梯度只是导数，或者，对于一个线性函数，也就是线的斜率。

梯度一词有时用于斜度，也就是一个曲面沿着给定方向的倾斜程度。可以通过取向量梯度和所研究的方向的点积来得到斜度。梯度的数值有时也被称为梯度。

所以寻找极小值也就是沿着梯度越来越低的方向不断搜寻，故而叫做梯度下降。

这里的梯度，就是前面提到的激活函数的导函数(不知道如何求导也没关系，因为就目前而言，常用的激活函数以及对应的导函数都可以在网上找到)，比如前向传播时，使用的激活函数是sigmoid (s型) 函数：

$$f(x) = \frac{1}{1 + e^{-x}}$$

其中：x：神经细胞所有输入（包括偏置输入）求和之后的值，f(x)：神经细胞的最终输出

其导函数为：

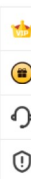
$$f(x)' = f(x)(1 - f(x))$$

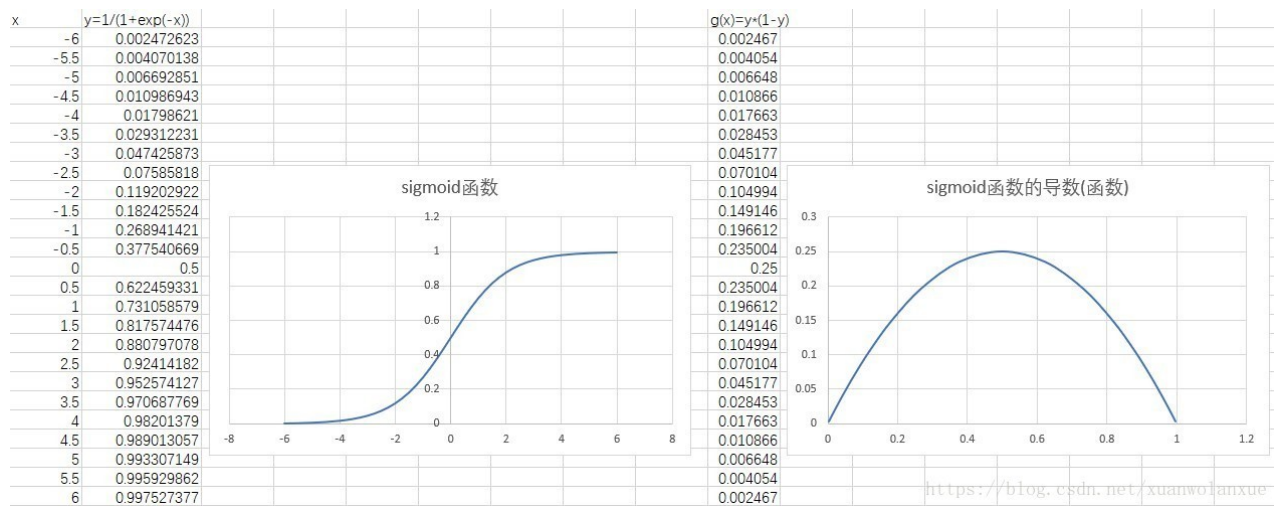
换一种形式就是：

$$f(x) = x(1 - x)$$

其中：x：神经细胞的输出；f(x)：当前神经细胞的梯度（参见上面对梯度的解释）

下图就是使用Excel绘制的sigmoid函数以及它的导数的波形图





通过上面的公式，就将当前神经细胞的输出反向传播到了当前神经细胞的输入侧，而当前神经细胞的每一个输入，代表的都是反向传播方向的下一层神经细胞层的一个神经细胞。要调整当前神经细胞的输入权重，就代表着，调整反向传播方向的下一层神经层的每一个神经细胞和当前神经细胞的连接权重。其调整幅度为：

$$\Delta W_{nj} = E_n LO_{(n-1)j}$$

其中：E：当前神经细胞的反向传播到输入侧的误差，L：学习率，O：前一层某个神经细胞的输出，n：代表第几层神经细胞，j：代表第几个输入或者反向传播方向的下一层中的第几个神经细胞。

对于上公式中的E，在不同的神经层的计算方法略微不同（主要是输出层和隐含层的不同）

其通用计算公式为：

$$E = e o(1 - o)$$

其中：e：神经细胞的输出误差，o(1 - o)：就是前面提到的使用sigmoid激活函数时神经细胞的梯度（f(x)=x(1-x)）

这里的差异就在于 e 的计算：

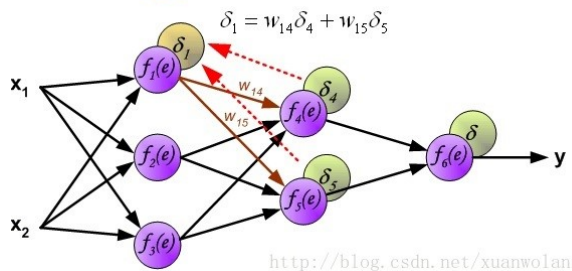
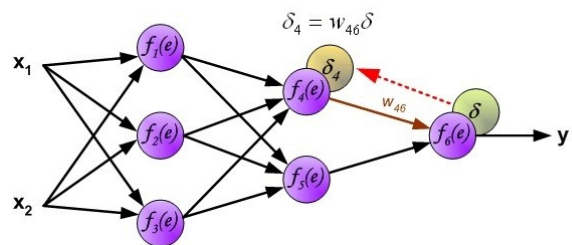
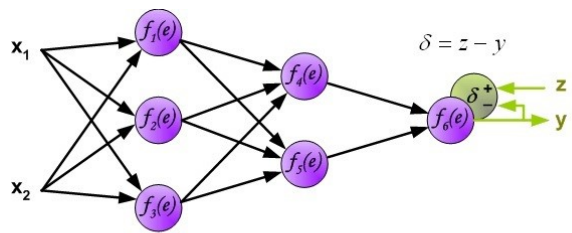
- 对于输出层：e = 当前神经细胞的输出 - 预期输出
- 对于隐含层：反向传播方向的上一层的所有神经细胞乘以与当前神经细胞的连接权重的累加和，其计算公式如下：

$$e = \sum_{j=0}^{j=k} e_{(n+1)j} w_{(n+1)j}$$

其中：j：表示反向传播方向的上一层的第几个神经细胞，n：表示第几层神经网络，n+1：表示在反向传播方向上当前层的上一层， $e_{(n+1)j}$ ：表示反向传播的上一层的第j个神经细胞的输出误差， $w_{(n+1)j}$ ：表示当前神经细胞与反向传播方向的上一层的第j个神经细胞的连接权重

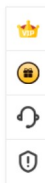
整体的反向传播算法的流程如下：

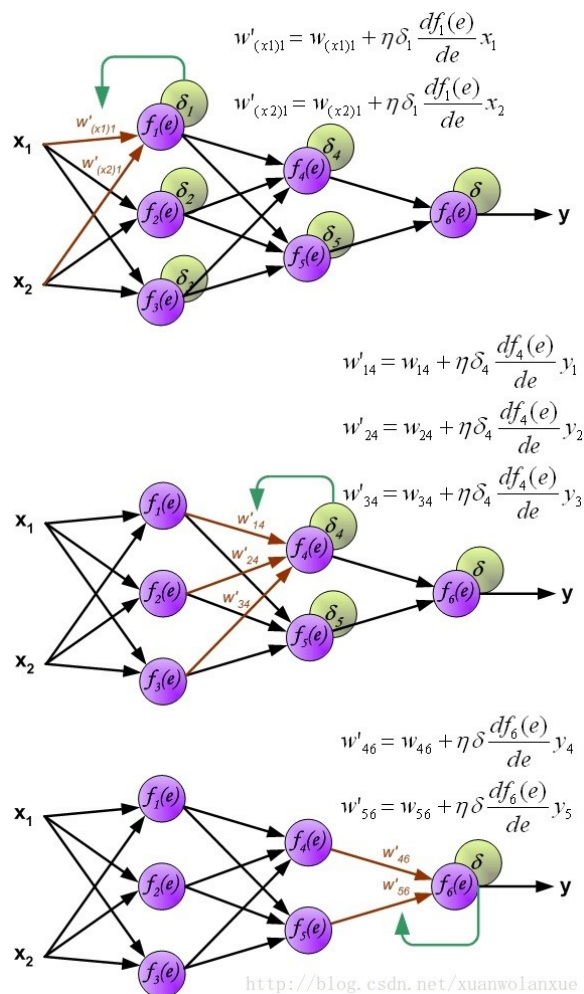
- 反向传播误差：



<http://blog.csdn.net/xuanwolanxue>

• 更新权重 (正向传播)





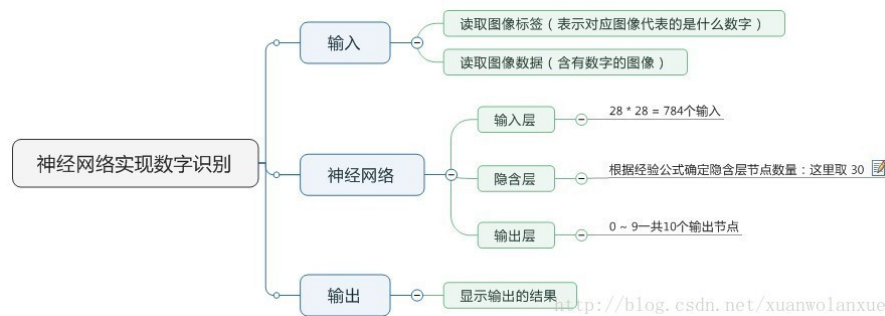
上图中, $\frac{df(e)}{de}$ 就是对激活函数求导, 如果使用的激活函数是sigmoid (s型函数), 那么其就可以用上面的 $f(x)=x/(1+x)$.

详细的信息 (Principles of training multi-layer neural network using backpropagation) 可以查看网站: http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

当然还有另一种反向传播方式, 和上面说到的不一样的地方是, 每一层神经细胞层反向传播误差之后就计算该层每个神经细胞包含的每个输入对应的权重, 然后再反向传播到下一层, 再重复权重变化的计算, 后面的例子当中就是使用的方法, 目的是减少循环次数, 加快训练速度。

三、手写数字识别

现在来说说如何使用神经网络实现手写数字识别。在这里我使用mind manager工具绘制了要实现手写数字识别需要的模块以及模块的功能:



其中隐含层节点数量 (即神经细胞数量) 计算的公式 (这只是经验公式, 不一定是最佳值):

$$m = \sqrt{n + l} + a$$

$$m = \log_2 n$$

$$m = \sqrt{nl}$$

- m: 隐含层节点数
- n: 输入层节点数
- l: 输出层节点数
- a: 1-10之间的常数

本例子当中:

- 输入层节点n: 784
- 输出层节点: 10 (表示数字 0 ~ 9)

隐含层选30个, 训练速度虽然快, 但是准确率却只有91% 左右, 如果将这个数字变为100 或是300, 其训练速度回慢一些, 但准确率可以提高到93%~94% 左右。

因为这是使用的MNIST的手写数字训练数据, 所以它的图像的分辨率是28 * 28, 也就是有784个像素点, 其下载地址为: <http://yann.lecun.com/exdb/mnist/>

这里输入和输出这两个模块都比较简单, 就不用讲了, 有兴趣的, 在文章末尾所提供的地址去下载源码便可以详细了解。主要还是讲解神经网络这一块。

3.1 神经细胞, 神经细胞层

要构建一个神经网络, 那么首先就需要构建它的基本单元, 神经细胞。根据前文讲到的内容, 一个神经细胞需要包含的内容为:

- 输入权重数组
- 输入数目
- 输出值
- 误差值

因为这里就是简单的bp (反向传播) 神经网络, 所以在同一个神经细胞层里面, 每一个神经细胞的输入数目都是相同的, 所以这里“输入数目”就可以省略掉, 以便节省内存。其大概形式如下:

```
1 struct neuron
2 {
3     //int mInputCount;    /** 当前神经细胞的输入数目 */
4     double mOutActivation; /** 当前神经细胞的输出 */
5     double mOutError;     /** 当前神经细胞的误差 */
6     vector<double> mWeights; /** 当前神经细胞的输入权重数组 */
7 };
```

同理推断神经细胞层的结构为:

```
1 struct neuronLayer
2 {
3     int mNumInputsPerNeuron; /** 当前层的每个神经细胞的输入数目 */
4     int mNumNeurons; /** 当前层的神经细胞数目 */
5     vector<neuron> mNeurons; /** 当前层的神经细胞数组 */
6 };
```

但是为了简化结构, 方便计算, 在本例子当中, 省略了神经细胞 (neuron) 这个结构, 直接定义神经细胞层, 在神经细胞层里面分别使用数组来存放神经细胞 (neuron) 结构里面的成员变量, 如下所示:

```
1 /** the type of neuron Layer */
2
3 struct neuronLayer
4 {
5 public:
6     neuronLayer(int numNeurons, int numInputsPerNeuron); /** 神经细胞层的构造函数 */
7
8     neuronLayer(neuronLayer& nl); /** 神经细胞层的拷贝构造函数 */
9     ~neuronLayer(); /** 神经细胞层的析构函数 */
10
11     void reset(void); /** 神经细胞层的重置函数 (将权重等参数都重置为随机值) */
12 public:
13     int mNumInputsPerNeuron; /** 当前层的每个神经细胞的输入数目 */
14     int mNumNeurons; /** 当前层的神经细胞数目 */
15     double** mWeights; /** 2维数组, 行: 代表神经细胞 (每一行就是一个神经细胞的所有权重), 列: 代表神经细胞的输入权重 */
16     double* mOutActivations; /** 当前层每个神经细胞的输出值 */
17     double* mOutErrors; /** 当前层每个神经细胞的误差值 */
18
19 };
```

有了神经细胞层, 接下来就该是实现神经网络了。

3.2 神经网络

本例子特征:



- 简单的全连接神经网络;
- 层数可自定义;
- 每一个隐含层的神经细胞数目可自定义;
- 使用反向传播以及梯度下降法进行网络演化;
- 激励函数使用: sigmoid (S型) 函数;
- 使用c++语言实现。

首先来看一下, 神经网络类的数据结构:

```

1 class bpNeuronNet
2 {
3 public:
4     bpNeuronNet(int numInputs, double learningRate); /** 构造函数 */
5     ~bpNeuronNet(); /** 析构函数 */
6 public:
7     /** 或者网络的总误差 (输出层每个神经细胞输出误差的方差总和) */
8     inline double getError(void) { return mErrorSum; }
9
10    bool training(const double inputs[], const double targets[]); /** 训练网络 */
11    void process(const double inputs[], double* outputs[]); /** 处理数据 (这里是数字识别) */
12
13    void reset(void); /** 重置网络 */
14    void addNeuronLayer(int numNeurons); /** 添加一个神经网络层 */
15
16 private:
17
18     /** 前向传播, 计算网络的输出 */
19
20     /** sigmoid(S型)激活函数 */
21     inline double sigmoidActive(double activation, double response);
22
23     /** 更新一个神经网络层, 计算其输出 */
24     void updateNeuronLayer(neuronLayer& nl, const double inputs[]);
25
26     /** 反向传播, 训练网络 */
27
28     /** 反向传播的激活函数, sigmoid函数的导数 */
29     inline double backActive(double x);
30
31     /** 以训练模式更新网络, 与 process 函数的区别是, 这个函数会更新输出层的每个神经细胞的输出误差 */
32     void trainUpdate(const double inputs[], const double targets[]);
33
34     /** 训练一层神经细胞层 */
35     void trainNeuronLayer(neuronLayer& nl, const double prevOutActivations[],
36                           double prevOutErrors[]);
37
38 private:
39     int mNumInputs; /** 神经网络的输入数目 */
40     int mNumOutputs; /** 神经网络的输出数目 */
41     /** 隐含层数目, 总的神经网络层的数目= mNumHiddenLayers + 1
42      * (不包含输入层, 因为它是直接映射到第一个隐含层, 没有神经细胞) */
43     int mNumHiddenLayers;
44     /** 神经网络的学习率 (表示学习速度的), 需要慎重选择, 太大会出现错误收敛或者无法收敛,
45      * 太小也可能导致错误收敛, 且学习速度变慢 */
46     double mLearningRate;
47     double mErrorSum; /** 忘了总误差, 参考: getError函数 */
48     vector<neuronLayer*> mNeuronLayers; /** 神经网络包含的神经细胞层数组 */
49 };

```

上面的类的成员函数的各个参数应该都能够自解释, 再加上中文注释, 应该就不需要再多说什么了。

注意: 因为输入层, 都是将数据直接映射到第一个隐含层, 所以, 它没有神经细胞, 这也是为什么在2.3节中三层神经网络结构图中, 输入层是绿色的正方形, 而不是后面两层那样的圆形。所以对于输入层不需要调用 addNeuronLayer 函数进行添加, 在构造神经网络时, 已经使用 numInputs 参数传入到神经网络了。

这里这些成员函数的具体实现:

正向传播方向的函数:

- 比较简单的 sigmoid (S型) 函数, 公式前面已经说过了, 代码实现如下:

```

1 double bpNeuronNet::sigmoidActive(double activation, double response)
2 { /** reponse 是用于缩放 activation的, 取值范围: 0 < response <= 1.0 */
3     /** sigmoid function: f(x) = 1 / (1 + exp(-x)) */
4     return (1.0 / (1.0 + exp(-activation * response)));
5 }

```

这里多了一个 response 参数, 其实它用来对当前神经细胞所有输入信号 (包括偏置) 的累加和进行缩放的, 前面2.4节中有关于 sigmoid 函数的波形图, 可以看到, x 的取值负的越小, 或者正的时候, 就越逼近 0 或 1 (可称他们为极值或者饱和区), 而且曲线也平缓, 根据前面说到的在训练时, 反向传播是 sigmoid 函数的导数, 也就是 sigmoid 曲线上的点相对应的斜率 (而这个斜率又是相应的梯度), 当逼近饱和区时, 因为曲线变得平缓, 斜率就趋近于 0, 这样对于得到的权重变化就会非常非常小, 也趋近于 0, 也就是几乎达不到演化神经网络的目的 (因为权重基本上没什么变化, 就没有调整权重的意义了), 所以调整这个值也可以改变学习度, 影响识别率, 有兴趣的可以试试。

- updateNeuronLayer函数，利用传入的inputs数据更新一个神经细胞层的输出数据：

```

1 void bpNeuronNet::updateNeuronLayer(neuronLayer& n1, const double inputs[])
2 {
3     int numNeurons = n1.mNumNeurons; /** 当前层有多少个神经细胞 */
4     int numInputsPerNeuron = n1.mNumInputsPerNeuron; /** 当前层的每一个神经细胞有多少输入 */
5     double* curOutActivations = n1.mOutActivations; /** 当前层所有神经细胞的输出值数组 */
6
7     /** 遍历每一个神经细胞 */
8     for (int n = 0; n < numNeurons; ++n)
9     {
10         double* curWeights = n1.mWeights[n]; /** 获取第n个神经细胞的输入权重数组 */
11
12         double netinput = 0;
13         int k;
14         /** 遍历每一个输入权重 */
15         for (k = 0; k < numInputsPerNeuron; ++k)
16         {
17             /** 累加 weights 和 inputs 的乘积 */
18             netinput += curWeights[k] * inputs[k];
19         }
20
21         /** 添加偏置项的值 */
22         netinput += curWeights[k] * BIAS;
23
24         /** 将累加后的值通过激活函数，得到当前神经细胞的最终输出 */
25         curOutActivations[n] = sigmoidActive(netinput, ACTIVATION_RESPONSE);
26     }
27 }
28 }
29

```

这个函数中有连个宏定义，ACTIVATION_RESPONSE是介绍sigmoidActive函数时已经说过的，这里不再多说，然后就是BIAS，这个就是偏置输入，可以取1或者-1，当取值为1时，其实这个宏就可以取消掉。

- process，数据处理函数，处理输入数据（本例是识别手写数字），得到相应输出

```

1 void bpNeuronNet::process(const double inputs[], double* outputs[])
2 {
3     /** 逐层更新网络 */
4     for (int i = 0; i < mNumHiddenLayers + 1; i++)
5     {
6         updateNeuronLayer(*mNeuronLayers[i], inputs);
7         inputs = mNeuronLayers[i]->mOutActivations;
8     }
9
10    /** 获取输出层的神经细胞的输出数组（即整个网络的最终输出结果）*/
11    *outputs = mNeuronLayers[mNumHiddenLayers]->mOutActivations;
12
13 }

```

此函数比较加单，就是正向的从：输入层（inputs参数）——>第一隐含层——>...——>输出层，逐层更新网络，并将每一层的结果馈送到下一层（前馈网络）。

以下就是反向传播方向（训练神经网络）的相关函数：

- backActive函数，之前说到的sigmoid函数的导数：

```

1 double bpNeuronNet::backActive(double x)
2 {
3     /**
4      *  $f(x) = x * (1 - x)$  sigmoid函数的导数
5      */
6     return x * (1 - x);
7 }

```

- trainUpdate，以训练模式更新网络的函数

```

1 void bpNeuronNet::trainUpdate(const double inputs[], const double targets[])
2 {
3     for (int i = 0; i < mNumHiddenLayers + 1; i++)
4     { /** 调用updateNeuronLayer函数，正向传播方向，循环的逐层更新网络 */
5         updateNeuronLayer(*mNeuronLayers[i], inputs);
6         inputs = mNeuronLayers[i]->mOutActivations;
7     }
8
9     /** 获取网络的输出层 */
10    neuronLayer& outLayer = *mNeuronLayers[mNumHiddenLayers];
11    double* outActivations = outLayer.mOutActivations; /** 输出层的神经细胞的输出数字 */
12    double* outErrors = outLayer.mOutErrors; /** 输出层的神经细胞的输出误差数组 */
13    int numNeurons = outLayer.mNumNeurons; /** 输出层的神经细胞数量 */
14
15    mErrorSum = 0; /** 重置整个网络的总误差 */
16    /** 更新输出层神经细胞的输出误差 */

```

```

17     for (int i = 0; i < numNeurons; i++)
18     {
19         //double err = outActivations[i] - targets[i];
20         double err = targets[i] - outActivations[i]; /** 获取误差 */
21         outErrors[i] = err; /** 保存误差 */
22         /** 更新方差累加和. (当这个值比预设的阈值小的时候, 就可以代表网络已经训练成功了) */
23         mErrorSum += err * err;
24     }
25 }

```

● trainNeuronLayer 训练一层神经细胞层

```

1 void bpNeuronNet::trainNeuronLayer(neuronLayer& nl, const double prevOutActivations[],
2                                   double prevOutErrors[])
3 {
4     int numNeurons = nl.mNumNeurons; /** 当前层的神经细胞数目 */
5     int numInputsPerNeuron = nl.mNumInputsPerNeuron; /** 当前层每个神经细胞的输入数目 */
6     double* curOutErrors = nl.mOutErrors; /** 当前层神经细胞的输出误差数组 */
7     double* curOutActivations = nl.mOutActivations; /** 当前层的神经细胞的输出数组 */
8
9     /** 遍历当前层的神经细胞, 并计算每个神经细胞的输出误差以及调整权重的依据 */
10
11     for (int i = 0; i < numNeurons; i++)
12     {
13         double* curWeights = nl.mWeights[i]; /** 获取当前神经细胞的输入权重数组 */
14         double coi = curOutActivations[i]; /** 获取当前神经细胞的输出 */
15         /** 利用反向传播激活函数计算反向传播回来的误差 */
16         double err = curOutErrors[i] * backActive(coi);
17
18         /** 遍历当前神经细胞的所有权重, 并基于反向传播回来的误差和学习率等参数计算新的权重值 */
19
20         int w;
21         /** 遍历当前神经细胞的权重, 不包括偏置项 */
22         for (w = 0; w < numInputsPerNeuron; w++)
23         {
24             /** 更新反向传播到, 反向传播方向的下一层相应神经细胞的输出误差 */
25             if (prevOutErrors)
26             { /** 因为输入层只有数据, 没有神经细胞, 所以此处需要判断此数组是否存在 */
27                 prevOutErrors[w] += curWeights[w] * err;
28             }
29
30             /** 基于反向传播规则计算新的权重 */
31             curWeights[w] += err * mLearningRate * prevOutActivations[w];
32         }
33
34         /** 更新当前神经细胞偏置项的权重 */
35         curWeights[w] += err * mLearningRate * BIAS;
36     }
37 }

```

trainNeuronLayer函数中计算新权重的规则处, "curWeights[w] += err * mLearningRate * prevOutActivations[w];", 这里使用的是 "+=", 这一点是需要和trainUpdate函数中 "double err = targets[i] - outActivations[i];" 相对应的, 也就是说当 "err = t - o;" 获取误差时, 需要使用 "+" (加等于), 当使用 "err = o - t;" 时, 需要使用 "-=" (减等于)

● training, 训练函数, 根据输入数据, 与预期结构对神经网络进行训练 (演化) :

```

1 bool bpNeuronNet::training(const double inputs[], const double targets[])
2 {
3     const double* prevOutActivations = NULL;
4     double* prevOutErrors = NULL;
5     trainUpdate(inputs, targets); /** 以训练模式更新网络 */
6
7     /** 以反向传播方向的顺序逐层训练网络 */
8     for (int i = mNumHiddenLayers; i >= 0; i--)
9     {
10         neuronLayer& curLayer = *mNeuronLayers[i]; /** 获取第i层神经细胞层 */
11
12         /** get the out activation of prev Layer or use inputs data */
13
14         if (i > 0)
15         {
16             /** 获取反向传播方向上的下一层神经细胞层 */
17             neuronLayer& prev = *mNeuronLayers[(i - 1)];
18             /** 获取反向传播方向上的下一层的神经细胞的输出数组 */
19             prevOutActivations = prev.mOutActivations;
20             /** 获取反向传播方向上的下一层的神经细胞的输出误差数组 */
21             prevOutErrors = prev.mOutErrors;
22             /** 重置获取反向传播方向上的下一层的神经细胞的输出误差为 "0" */
23             memset(prevOutErrors, 0, prev.mNumNeurons * sizeof(double));
24
25         }
26         else
27         {
28             /** i=0时, 表示第一层隐含层, 它的输入就是整个网络的输入数据 (即所谓的输入层) */

```

```

29         prevOutActivations = inputs;
30         prevOutErrors = NULL;
31     }
32
33     /** 调用trainNeuronLayer函数训练第i层神经细胞 */
34     trainNeuronLayer(curLayer, prevOutActivations, prevOutErrors);
35 }
36
37 return true;
38 }

```

自此，神经网络的正向和反向的相关成员函数都已介绍完毕，整体上来说，代码不多，理解了原理，也就不难理解这些代码了，接下来就是介绍神经网络识别手写数字的测试程序（包括训练和测试识别）

3.3 神经网络识别手写数字测试程序

不说废话，直接进入主题。首先来看训练函数：

```

1 double trainEpoch(dataInput& src, bpNeuronNet& bpnn, int imageSize, int numImages)
2 {
3     double net_target[NUM_NET_OUT]; /** 存放网络的预期输出 */
4     char* temp = new char[imageSize]; /** 创建临时数组来存放读取的手写数字的图像 */
5     progressDisplay progd(numImages); /** 显示进度的工具类 */
6
7     double* net_train = new double[imageSize]; /** 存放网络输入数据 */
8     for (int i = 0; i < numImages; i++)
9     {
10         int label = 0;
11         memset(net_target, 0, NUM_NET_OUT * sizeof(double));
12
13         if (src.read(&label, temp)) /** 读取一张手写数字的图像和代表这个图像中数字的标签 */
14         {
15             net_target[label] = 1.0; /** 将标签对于的预期输出位置 1 */
16
17             /**预处理手写数字图像数据为网络输入数据，并添加相应的噪声 */
18             preProcessInputDataWithNoise((unsigned char*)temp, net_train, imageSize);
19             /** 训练网络 */
20             bpnn.training(net_train, net_target);
21
22         }
23         else
24         {
25             cout << "read train data failed" << endl;
26             break;
27         }
28         //progd.updateProgress(i);
29
30         progd++; /** 更新训练进度 */
31     }
32
33     cout << "the error is:" << bpnn.getError() << " after training " << endl;
34
35     delete []net_train;
36     delete []temp;
37
38     return bpnn.getError();
39 }

```

上面函数中，预处理手写数字图像数据时，之所以加入噪声，是为了防止过拟合（至于什么是过拟合，还请问度娘去）的一种技巧，添加摇摆（jitter）。《游戏编程中的人工智能》一书的第九章第三节有讲到。

然后是测试识别函数：

```

1 int testRecognition(dataInput& testData, bpNeuronNet& bpnn, int imageSize, int numImages)
2 {
3     int ok_cnt = 0;
4     double* net_out = NULL;
5     char* temp = new char[imageSize]; /** 创建临时数组来存放读取的手写数字的图像 */
6     progressDisplay progd(numImages); /** 显示进度的工具类 */
7     double* net_test = new double[imageSize]; /** 存放网络输入数据 */
8     for (int i = 0; i < numImages; i++)
9     {
10         int label = 0;
11
12         if (testData.read(&label, temp)) /** 读取一张手写数字的图像和代表这个图像中数字的标签 */
13         {
14             /** 预处理手写数字图像数据 */
15             preProcessInputData((unsigned char*)temp, net_test, imageSize);
16
17             /**利用神经网络处理（识别） 手写数字图像数据 */
18             bpnn.process(net_test, &net_out);
19
20
21             /** 遍历神经网络的所有输出，找出最大的一个 */
22             int idx = -1;

```

```

23         double max_value = -99999;
24         for (int i = 0; i < NUM_NET_OUT; i++)
25         {
26             if (net_out[i] > max_value)
27             {
28                 max_value = net_out[i];
29                 idx = i;
30             }
31         }
32
33         if (idx == label)
34         {/** 如果最大输出对应的神经细胞的idx与读取的数字对应的标签相等，表示识别成功，计数加1 */
35             ok_cnt++;
36         }
37
38         progd.updateProgress(i); /** 更新识别进度 */
39     }
40     else
41     {
42         cout << "read test data failed" << endl;
43         break;
44     }
45 }
46
47
48 delete []net_test;
49 delete []temp;
50
51 return ok_cnt;
52
53 }

```



20



26



以上函数中，之所以要遍历神经网络的输出，找到最大的一个，是因为我们使用的激活函数（sigmoid函数）：

$$f(x) = \frac{1}{1 + e^{(-x)}}$$

要得到1，必须x取正无穷大，这显然不现实。所以只能找最接近1的一个输出，作为网络的有效输出。

最后简单的看一下main函数：

```

1  int main(int argc, char* argv[])
2  {
3      dataInput src; /** 用于训练神经网络的数据读取对象 */
4      dataInput testData; /** 用于测试识别的数据读取对象 */
5      bpNeuronNet* bpnn = NULL;
6      srand((int)time(0));
7
8      if (src.openImageFile("train-images.idx3-ubyte") &&
9          src.openLabelFile("train-labels.idx1-ubyte"))
10     {/** 打开手写数字图像数据文件以及对应的标签文件 */
11         int imageSize = src.imageLength(); /** 获取一张手写数字图像的大小（像素总和） */
12         int numImages = src.numImage(); /** 获取数据集有多少张图像 */
13         int epochMax = 1;
14
15         double expectErr = 0.1;
16
17         bpnn = new bpNeuronNet(imageSize, NET_LEARNING_RATE); /** 创建神经网络 */
18
19         /** add first hidden layer */
20
21         bpnn->addNeuronLayer(NUM_HIDDEN); /** 添加隐含层 */
22
23         /** add output layer */
24         bpnn->addNeuronLayer(NUM_NET_OUT); /** 添加输出层 */
25
26         cout << "start training ANN..." << endl;
27
28         /** 训练网络 */
29         for (int i = 0; i < epochMax; i++)
30         {
31             double err = trainEpoch(src, *bpnn, imageSize, numImages);
32
33             //if (err <= expectErr)
34             {
35                 // cout << "train success,the error is: " << err << endl;
36                 // break;
37             }
38
39             src.reset();
40         }
41
42         cout << "training ANN success..." << endl;
43
44         showSeparatorLine('-', 80);
45     }

```



```

46     if (testData.openImageFile("t10k-images.idx3-ubyte") &&
47         testData.openLabelFile("t10k-labels.idx1-ubyte"))
48     {
49         /** 打开用于测试识别的手写数字图像数据以及相应的标签数据 */
50         imageSize = testData.imageLength(); /** 获取一张手写数字图像的大小 (像素总和) */
51         numImages = testData.numImage(); /** 获取数据集有多少张图像 */
52
53         cout << "start test ANN with t10k images..." << endl;
54
55         /** 测试识别 */
56         int ok_cnt = testRecognition(testData, *bpnn, imageSize, numImages);
57
58         cout << "digital recognition ok_cnt: " << ok_cnt << ", total: " << numImages << endl;
59     }
60     else
61     {
62         cout << "open test image file failed" << endl;
63     }
64
65 }
66 else
67 {
68     cout << "open train image file failed" << endl;
69 }
70
71 if (bpnn)
72 {
73     delete bpnn; /** 销毁神经网络 */
74 }
75
76 getch();
77
78 return 0;
79 }

```

注意: addNeuronLayer函数, 必须按照正向传播方向的顺便添加, 即 第一隐层——>...——>第n隐层——>输出层, 这样的顺序。

四、总结与后续

神经网络的原理虽然看上去不难, 但实现起来还是不太简单, 这还最简单的神经网络, 如果是CNN, RNN, 深度网络等等那就更加复杂了, 且参数之多, 也是难以想象的。

第三章的例子还是只比较粗糙的版本, 后续还可以添加更多的功能, 比如:

- 将宏定义和网络层数等进行参数化, 外部使用ini文件之类的配置文件, 测试程序读取这些配置就可以构建和初始化神经网络。
- 在神经网络类中添加成员函数来提取和设置所有神经细胞的输入权重, 这样一个训练好的神经网络, 就可以将权重提取出来, 保存为一个配置文件, 下次就可以直接读取配置文件来初始化权重, 这样就不需要每次使用的时候都要先对网络进行演化。
- 使用遗传算法来演化网络中神经细胞的权重, 然后再使用反向传播来训练网络。
- ...

一个简单的神经网络的基本知识就是这些了, 下一步就需要研究研究 深度学习之类的东东了。

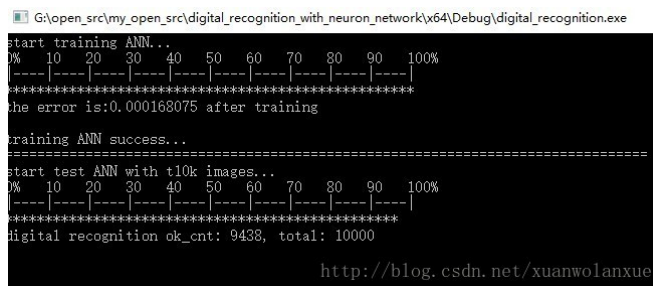
五、完整代码

完整代码时放在开源中国上面, 有兴趣的可以抓取来瞧一瞧。其地址如下:

https://gitee.com/xuanwolanxue/digital_recognition_with_neuron_network.git

六、更新

添加上程序运行之后的识别结果, 如下图所示:



```

G:\open_src\my_open_src\digital_recognition_with_neuron_network\64\Debug\digital_recognition.exe
start training ANN...
0% 10 20 30 40 50 60 70 80 90 100%
|-----|
*****
the error is:0.000168075 after training

training ANN success...
=====
start test ANN with t10k images...
0% 10 20 30 40 50 60 70 80 90 100%
|-----|
*****
digital recognition ok_cnt: 9438, total: 10000

http://blog.csdn.net/xuanwolanxue

```

从上图可以看出, 使用10000个样本, 最后正确识别的为9438个, 正确识别率为94.38%。

[2018年8月7日更新]: 在原有的基础上增加了训练和测试各自的用时时间, 其结果如下所示:

```
G:\opt\digital_recognition_with_neuron_network\x64\Debug\digital_recognition.exe
start training ANN...
0% 10 20 30 40 50 60 70 80 90 100%
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
*****
the error is:0.00040444 after training

training ANN success...cast time: 89714(millisecond)
*****
start test ANN with t10k images...
0% 10 20 30 40 50 60 70 80 90 100%
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
*****
Digital recognition cast time:6485(millisecond), ok_cnt: 9424, total: 10000

https://blog.csdn.net/xuanwolanxue
```

👍
20

🔗

💬
26

📖

🔖

🔍

🏆

有 0 个人打赏

文章最后发布于: 2017-05-10 18:20:42

股神狱中曝出庄家洗盘规律，牢记这3点，看完恍然大悟

股管家 · 顶新

👤 想对作者说点什么

👤 是大大环呀 ~ 4个月前 #8楼

查看回复(4)

👍

博主你好，我使用进过处理的mnist图片进行试验，但是每次正确率都只有10%，等于没有测试随机分配结果，请问这是什么情况呢，烦请解答一下。

👤 流年Bast 4个月前 #7楼

查看回复(6)

👍

老哥，我这边打开工程有一堆的报错，比如main.cpp中的无法打开源文件“time.h”，还有bpNeuronNet里的training函数中的size_t不明确，我看你文章中没有这个参数，不太明白，希望大佬解惑

👤 weixin_42936803 1年前 #6楼

查看回复(2)

👍

大神，请教一下，我如想要把激活函数改成ReLU，除了修改sigmoidActive，backActive这两个函数外，还需要在哪里做改动？谢谢！

👤 a641864291 9个月前 #5楼

查看回复(1)

👍

开源中国上代码的地址显示页面不存在是为什么呢，

👤 登录 查看 26 条热评

- 用python创建的神经网络--mnist手写数字识别率达到98%

阅读量 7503

周末根据TariqRashid大神的指导，没有使用tensorflow等框架，用python编写了一... 博文 来自: 学习机器...
- Python代码实现简单的MNIST手写数字识别（适合初学者看）

阅读量 2万+

补充：由于很多同学找我我要原数据集和代码，所以我上传到了资源里，https://downl... 博文 来自: zugexiao...
- 神经网络初识——MINIST手写数字识别（可视化输出结果）

阅读量 1万+

手写数字识别是每个学习神经网络的人上手操作的必由之路，今天在前人肩膀上做了... 博文 来自: Finley199...
- 基于神经网络的手写数字识别（上）

阅读量 1427

导读：近期人工智能火热，十分好奇，决定了解一些基础知识。本篇博客将从一个数... 博文 来自: qq_1568...
- 股票“三不卖七不买”1定律分析法，准确率惊人！

巨量 · 顶新
- Tensorflow卷积神经网络（CNN）手写数字识别示例学习

阅读量 1万+

目录一、问题描述二、数据描述三、网络结构1.输入层2.卷积层3.池化层4.全连接层5.... 博文 来自: 一颗贪婪...
- 机器学习Tensorflow基于MNIST数据集识别自己的手写数字（读取和测试...

阅读量 3万+

废话不多说，先上效果图整体来看，效果是非常不错的，模型的训练，参照官方代码... 博文 来自: qq_3826...
- python读取MNIST数据集

阅读量 7109

在学习ufidl课程时需要用到MNIST数据集，主页在这里。但由于该数据集为IDX文件... 博文 来自: jiede1的...
- 手写数字识别

阅读量 198

前面的博客介绍过神经网络结构以及相关的损失函数，在这里我们通过一个简单的... 博文 来自: God_68...
- CNN实现MNIST手写数字识别

阅读量 1万+

关键词：CNN、TensorFlow、卷积、池化、特征图一.前言本文用TensorFlow实现了... 博文 来自: lijiecao02...
- ...神经网络的优化(c++版本) - xuanwolanxue的专栏 - CSDN博客...

👑

🏆

🔄

🔔

TensorFlow学习---实现mnist手写数字识别 - huahuazhu..._CSDN博客

10-5

女孩子千万不要让男票发现这传奇！开局一条龙吸引力太大了！

贪玩游戏 · 顶新

机器学习：手写数字识别(Hand-written digits recognition)小项目

阅读数 7917

该项目的所有代码在我的github上，欢迎有兴趣的同学与我探讨研究~地址：Machin... 博文 来自： linwh8的...

神经网络初识——MINIST手写数字识别(可视化输出结果) ..._CSDN博客

9

三层神经网络实现手写数字的识别(基于tensorflow) - DL..._CSDN博客

11

My Machine Learn(四)：mnist数字识别神经网络的优化(c++版本)

阅读数 3617

一、背景去年写过一篇关于用c++实现mnist手写数字识别的神经网络的文章，当然，... 博文 来自： xuanwola...



周雄伟

113篇文章

关注

排名:千里之外



足各小兒

7篇文章

关注

排名:千里之外



Finley1991

2篇文章

关注

排名:千里之外



KEENE s

9篇文章

关注

排名:千里之外

...你搭建BP神经网络,并实现手写mnist手写数字识别 - w..._CSDN博客

4-30

基于神经网络的手写数字识别(上) - qq_15689151的博客 - CSDN博客

10-4

BP神经网络识别手写数字项目解析及matlab实现

阅读数 1万+

BP神经网络指传统的人工神经网络，相比于卷积神经网络(CNN)来说要简单些。人工... 博文 来自： ywxk131...

Python(TensorFlow框架)实现手写数字识别系统

阅读数 7万+

本文使用Tensorflow框架进行Python编程实现基于卷积神经网络的手写数字识别算法... 博文 来自： louishao...

深度学习(一)——简单神经网络识别手写数字 - 蘇ゝ的博..._CSDN博客

11-15

单层神经网络实现mnist手写数字识别 - wangTongGen - CSDN博客

4-8

C++从零实现深度神经网络之六——实战手写数字识别 (sigmoid和tanh)

阅读数 1万+

本文由@星沉阁冰不语出品，转载请注明作者和出处。文章链接：http://blog.csdn.n... 博文 来自： 星沉阁

股神狱中曝出庄家洗盘规律，牢记这3点，看完恍然大悟

股管家 · 顶新

tensorflow简单全连接神经网络,识别minist手写数字 - a..._CSDN博客

11-30

DNN实现MNIST手写数字识别 - 清闲老窝的专栏 - CSDN博客

10-4

TensorFlow学习---实现mnist手写数字识别

阅读数 1819

卷积神经网络CNN的理论部分直接看帖子：TensorFlow学习--卷积神经网络CNN卷... 博文 来自： huahuaz...

手写数字识别的几种实现方法

阅读数 1万+

我使用了手写数字数据库MNIST的一个子集，并做了多种处理程序中附有数据库，完... 博文 来自： onezeros...

MNIST | 基于k-means和KNN的0-9数字手写体识别

阅读数 2330

本文介绍基于k-means和KNN的0-9数字手写体识别，把k-means聚类 and CNN识别应... 博文 来自： Chen_Tia...

C++实现神经网络识别数字

阅读数 1408

0.综述我用的神经网络是ANN，下文会介绍训练ANN的反向传播算法并给出相应的数... 博文 来自： GaoJieVe...

使用mnist数据集实现手写数字识别

阅读数 1765

mnist数据集中数字是0到9，要求实现多分类，需要使用softmax函数。此次实现单... 博文 来自： 第二剑~...

股王经典口诀：“换手率”涨三不追，跌四不压...看完恍然大悟！

巨景 · 顶新

【深度学习】python实现简单神经网络以及手写数字识别案例

阅读数 1521

前言\quad\quad为了能够解决感知机人工设定权重的工作，即确定合适的、能符合预... 博文 来自： Daycym...

【MATLAB】BP神经网络识别MNIST手写数字

阅读数 9059

1.Summary本文运用BP神经网络对MNIST手写数字字符进行识别。BP神经网络是通... 博文 来自： github_3...



20



26



9



11

