MNIST前馈神经网络的实现

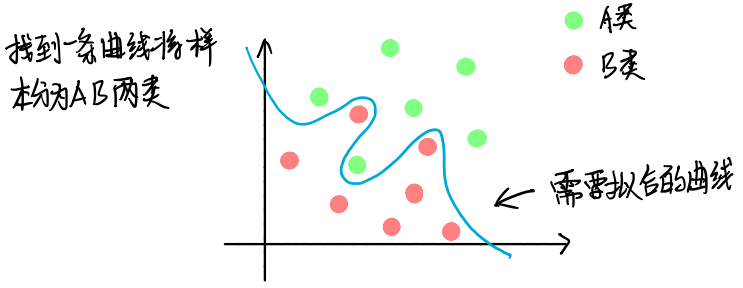
Written by wnistol

1.什么是神经网络?

神经网络是一个由节点（又名神经元）和边构成的网状结构，因结构和人脑神经相似而被称为神经网络。

2.神经网络有什么用?

神经网络的本质是一个函数拟合器，做的工作是用多项式拟合一个函数曲线（曲面），在分类任务中神经网络需要拟合一个曲线（曲面）来将输入的数据分类，例如识别手写数字0～9也就是将输入的手写数字图像分为0～9这十类，下面是一个简单的分类案例：

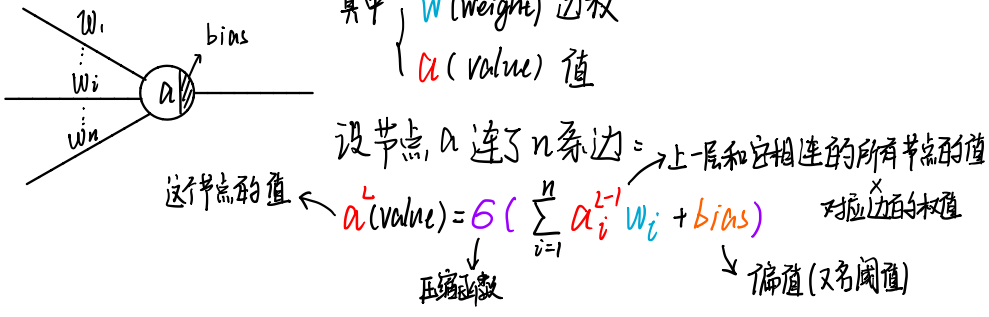


为了知道是否成功拟合这条曲线，首先定义一个误差函数E(x)来表达在给神经网络不同输入时神经网络输出值f(x)和理想曲线的函数g(x)输出值的误差。当在所有样本输入时输出值和理想值的误差足够小时，就得到了一个可以用来进行分类的可用的神经网络。换而言之，对于神经网络的训练，其目标也是找到E(x)在所有输入中的最小值（即最小化输出值和理想值之间的误差）。

3.神经网络的结构

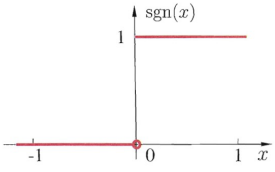
神经网络的基本组成单位是节点（神经元）和层（layer），每一层有多个节点，前馈神经网络中层与层之间每个节点相互连接构成一个神经网络。

其中神经元的结构如下：



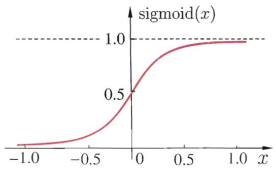
其中压缩函数（激活函数）是一个非线性函数，负责将上一次的值映射到这个节点上，其非线性函数的特性保证了神经网络可以拟合非线性函数（除去压缩函数外所有的操作都是矩阵相乘（线性）固无法拟合一个非线性函数）。

常用的压缩函数有：



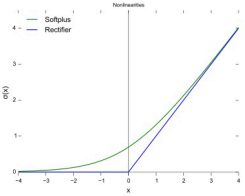
$$\mathrm{sgn}(x) = \begin{cases} 1, & x \geqslant 0; \\ 0, & x < 0. \end{cases}$$

(a) 阶跃函数



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

(b) Sigmoid 函数



Plot of the rectifier (blue) and softplus (green) functions near $x = 0$

ReLU（线性整流函数）

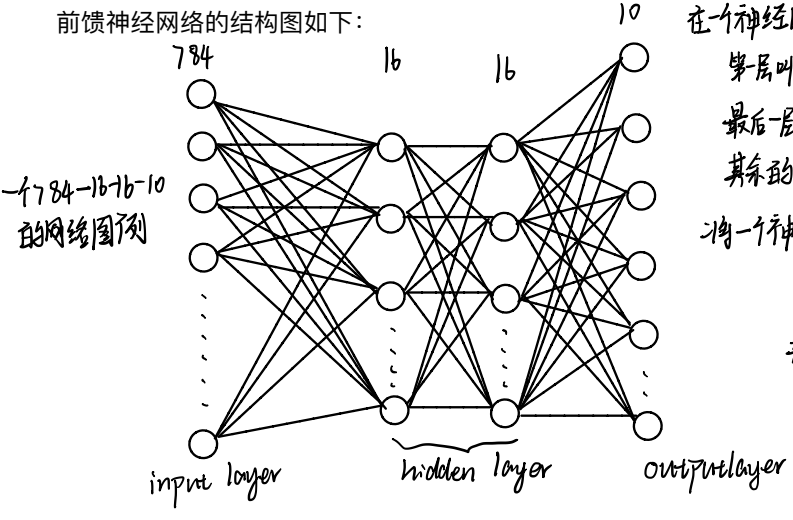
对于每个节点，有一个常量bias（偏置）又名 阈值，由此可得计算每一个节点上的值时需要：

1.计算每一个和该节点相连的上一层节点的值乘以对应边的权重并求和
2.将上面求和得到的结果加上偏置
3.将以上值塞进压缩函数计算映射后的值

前馈神经网络中，节点与节点间的连接方式是由一个叫“层（layer）”的方式连接的，连接方式满足：

1.同层之间节点互不相连
2.相邻两层的节点相互连接
3.节点与节点之间不能跨层连接

前馈神经网络的结构图如下：



在一个神经网络中：

第一层叫：输入层（input layer）

最后一层叫：输出层（output layer）

其余的中间层叫：隐藏层（hidden layer）

将一个神经网络视为一个函数：

$y = f(\vec{x})$ 对输入的处理（黑箱）（隐层）

$\vec{x}$ 输入层　y 输出层

4.识别手写数字的神经网络是如何工作的

如上文所述，神经网络的本质是拟合一个理想函数，所以该函数的执行过程就是从输入层开始计算从输入层到输出层的所有节点的值最后到输出层每个节点的值就是网络的输出值，这个过程叫做正向传播（Forward Porpagation）

在这个网络中输入层为手写数字的图片，图片尺寸为28x28总共748个像素点，由于是黑白图像固每个像素点的值只有一个，所以这个网络中我们将每个像素的值作为输入层对应节点的值，输入层设置为784个节点。

而我们要识别数字0～9固将输出层设置为10个节点，每个节点分别表示该图片是0到9的概率。

所以由样本情况和事件需求可以设定好输入层和输出层的参数，至于隐藏层部分将其设置为2层每层16个节点，至于为什么是2层16是我瞎tm说的，节点数和层数越多网络的拟合能力就越强（多项式的项数更多），但同样的运行速度会更慢也更吃内存，至于多少是合适的隐藏层参数可以通过自己实验或者参考MNIST官网的其他人写的神经网络参数。

5.如何训练一个前馈神经网络

前面提到需要建立一个函数E(x)来表达神经网络的实际输出和理想输出的误差从而反应该网络对目标函数的拟合情况，这样的函数叫做损失函数（loss function），在这里我们用误差的平方来作为这个损失函数，即：

$$E(x) = (y - \hat{y})^2$$

（y：实际输出；$\hat{y}$：理想输出）

这样我们就得到了一个输出值为实际和理想差距的函数（至于为什么用平方可以理解为将误差放大的作用）这样以来，只要找到合适权值和偏置能够让这个函数的输出值尽量小，我们就可以得到一个拟合得尽量好的曲线，此时，训练一个神经网络的目标就变成了寻找误差函数的最小值。

至于如何找到误差函数的最小值，使用了一个叫做梯度下降（Gradient Descent）的方法，众所周知函数在某点的梯度即为函数值变化最快的方向，而沿着梯度方向下降则是误差函数值下降最快的方法，使用梯度下降的方法对神经网络进行权值和偏置调整的方法叫做反向传播法（Backward Porpagation）。

6.反向传播法的实现

1.反向传播法的原理：

反向传播法通过链式法则计算每条边的权重和每个偏置对误差函数输出的倒数（函数的输出值对每个权值和偏置的敏感度）来确定如何调整每条边的权值以及每个节点的偏置：

$$\Delta w_{ij}^{L} = \eta \times \frac{\partial E_k}{\partial w_{ij}^{L}}$$ （w_{ij}^{L}：第L层，i点到j点的边；η：学习率；E_k：损失函数）

$$\Delta bias_j^{L} = \eta \times \frac{\partial E_k}{\partial bias_j^{L}}$$ （$bias_j^{L}$：第L层第j个点；η：学习率；E_k：损失函数）

所以，对于每条要调整的边和权重，都要求出上面对应的值，下面来分析一下如何求得这两个值（学习率为自己初始设定）：

设节点 a_j^L 的：值 $= a_j^L$；压缩函数 $= f(x)$；压缩前的值 $= z_j^L$

则有：$\frac{\partial E_k}{\partial w_{ij}^{L}} = \frac{\partial E_k}{\partial a_j^L} \times \frac{\partial a_j^L}{\partial z_j^L} \times \frac{\partial z_j^L}{\partial w_{ij}^L}$（链式法则）

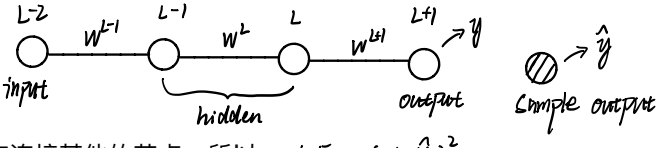
已知：$a_i^L = f(z_j^L)$，$z_j^L = \sum_{i=1}^{n}(a_i^{L-1} w_{ij}^{L}) + bias_j^{L}$（$n$：上层节点数；$i=1$：上一层所有与 a_j^L 相连的节点；w_{ij}^L：上一层到此节点对应的边；$bias_j^L$：上一层到此节点的值）

可得：$\frac{\partial a_j^L}{\partial z_j^L} = f'(z_j^L)$，$\frac{\partial z_j^L}{\partial w_{ij}^L} = a_i^{L-1}$

$$\therefore \frac{\partial E_k}{\partial w_{ij}^{L}} = \frac{\partial E_k}{\partial a_j^L} \times f'(z_j^L) \times a_i^{L-1}$$

同理可得：$\frac{\partial E_k}{\partial bias_j^L} = \frac{\partial E_k}{\partial a_j^L} \times f'(z_j^L) \times 1$

从上式可以看出，更新一条边的权值首先需要得到这条边所属点的偏导 $\frac{\partial E_k}{\partial a_j^L}$ 那么，如何求得 $\frac{\partial E_k}{\partial a_j^L}$ 的值呢？首先我从一个最简单的每一层只有一个节点的网络入手：

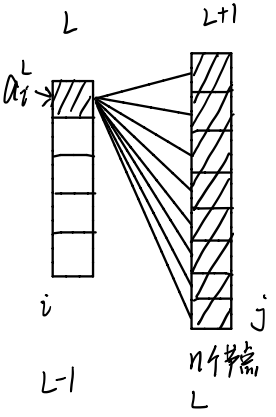


对于输出层，他的后面没有连接其他的节点，所以：

$$\because E_k = (y-\hat{y})^2$$
$$\therefore E_k = (a^{L+1}-\hat{y})^2$$
$$\therefore \frac{\partial E_k}{\partial a^{L+1}} = 2(a^{L+1}-\hat{y})$$

而对于输出层的上一层：$\frac{\partial E_k}{\partial a^L} = \frac{\partial E_k}{\partial a^{L+1}} \times \frac{\partial a^{L+1}}{\partial a^L}$ 其中：$a^{L+1} = a^L \times w^{L+1} \Rightarrow \frac{\partial a^{L+1}}{\partial a^L} = w^{L+1}$

同理，对于L层的上一层则有：$\frac{\partial E_k}{\partial a^{L-1}} = \frac{\partial E_k}{\partial a^L} \times \frac{\partial a^L}{\partial a^{L-1}}$ 由此可知前一层的更新依赖上一层的偏导值

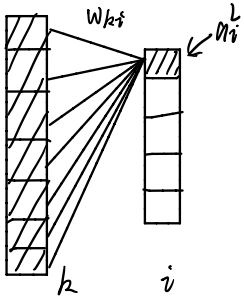


同理，对于上一层有n个节点的情况，可以知道更新aL的偏导需要对上一层中每个与aL相连的节点的偏导，即相应边的权值：

$$\frac{\partial E_k}{\partial a_i^L} = \sum_{j=1}^{n}\left(\frac{\partial E_k}{\partial a_j^{L+1}} \times W_{ij}^{L+1}\right)$$

← 更新 $\frac{\partial E_k}{\partial a_i^L}$

依赖这一层和下一层



由上面公式计算出aL的偏导以后就可以根据前面提到的 压缩函数的导数和上一层对于节点的值来计算与aL相连的每条边的权重的偏导与aL节点偏置的偏导了。至此，神经网络的常用训练方法反向传播法就已经讲解完成，除此之外还有许多其他的训练方法譬如模拟退火和遗传算法我也还没学，有兴趣的朋友可以去看看然后带带我（QAQ）

← 更新 $\frac{\partial E_k}{\partial W_{ki}^L}$ 和 $\frac{\partial E_k}{\partial bias_i^L}$

本项目中用到的 Sigmoid 导数为：

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

依赖这一层和上一层

7.总结

在前面的内容中已经提到了神经网络的本质是一个函数，对其的训练是让网络能够拟合目标函数，所以使用神经网络的操作叫做正向传播，即从输入层开始对每一层计算每个节点的值直到输出层为止，对神经网络进行训练的方法叫做反向传播，即计算损失函数的输出对每个权重及偏置的偏导从而调整权重和偏置以达到拟合目标函数的目的。

8.用到的其他算法：

1.随机数产生：Multiply With Carry Algorithm
2.初始化权重：Nguyen-Widrow Algorithm

推荐配合3b1b的视频食用：https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi