# 基于混合神经网络的MINST识别算法

## 功能介绍

本算法旨在通过用利用神经网络的优势，高效的对手写数字进行识别，即将手写在纸上或设备上的数字的图片信息高正确率地转换为纸上或设备上的数字。

## 整体架构介绍

为了充分发挥各个神经网络的优点，使其既有卷积神经网络高效，速度快，对噪点的容错性高的优点，又有全联接的前导神经网络准确性高的特点，本算法使用将卷积神经网络与前导神经网络抽样后全联接的方法来高效的完成对手写数字图像的识别。

>整体网络图 figure\_all

本神经网络的主体分为输入数据处理，神经网络运算和输出数据处理。

其中输入数据处理（IN\_PRO）是将原始数据处理成神经网络容易识别的且经过标准化的数据集。

神经网络(在运算上分为训练阶段与识别阶段。在训练阶段神经网络通过所给的训练数据与答案修改与更新自己的参数，从而使自己通过输入数据所计算出来的答案越来越接近真实正确的答案。识别阶段即使用通过大量数据训练过的神经网络，让测试数据通过神经网络进行运算从而得出预测的结果。

在这里神经网络部分（NN）分为输入层（IN）卷积层（C1），池化层（S1），隐藏层（H1），输出层（OUT）

输出处理则将大量的原始，为优化结果而标准化的数据处理成可以利用，统计的直观数据。在这里输出处理只有一部分（OUT\_PRO）。

## 详细介绍

下面将对本算法的各个层次的原理，设计与实现进行详尽的介绍。

### 输入数据处理部分

>输入处理图 in1

本部分旨在对原始数据的处理，其中原始数据有训练数据（32000张PNG图片，其中每个图片的分辨率为28\*28，图片为灰度图片）；训练数据的标签，即这32000张图片所显示的具体数字的答案；测试数据（10000张图片，图片格式与训练数据相同）以及测试数据的标签。

对训练与测试数据的处理：

训练数据原始为图片，先用python的Image库将整个图片加载成矩阵，由于输入的图片均为灰阶图片，所以根据Image库的算法，输出的矩阵为输入图片对应像素的灰阶值。

然后对这个矩阵进行标准化，标准化的意思是，通过线性或者非线性转换，将矩阵的最大值-最小值范围控制在0-1之间且让最大值和最小值尽量分别接近0与1。对于本文针对的灰阶矩阵，其最大值为255，最小值为0，所以我们可以通过把每个数字除以255的方式将其标准化。

用此方法，我们成功将输入矩阵进行了标准化。对于标准化的作用，本文将在后面介绍线性神经元与其梯度下降算法的部分详细介绍。

对训练与测试标签数据的处理：

训练数据与测试数据通过神经网络的计算会得到其预测的结果的特征，在这里这个特征的值就决定了本神经网络对输入图像的识别结果。区别于数字本身的连续性，对于神经网络的分类任务来说，其特征应该是互不相干的，例如：1与9之间的相关性应和1与2之间的相关性完全一样。所以需要用编码的方式排除数字本身的相关性，本算法使用的编码方式是独热编码（One-Hot Encoding）

独热编码：

独热编码直观来说就是用总状态数的比特数，其中一个比特为0其他比特为1的方法来表示状态的一种码制。

在本文的算法中，总状态数即为10。下面是编码表：

|  |  |  |
| --- | --- | --- |
| 原始数据 | 二进制 | 独热编码 |
| 0 | 0000 | 1000000000 |
| 1 | 0001 | 0100000000 |
| 2 | 0010 | 0010000000 |
| 3 | 0011 | 0001000000 |
| 4 | 0100 | 0000100000 |
| 5 | 0101 | 0000010000 |
| 6 | 0110 | 0000001000 |
| 7 | 0111 | 0000000100 |
| 8 | 1000 | 0000000010 |
| 9 | 1001 | 0000000001 |

表格 1-a

经过编码后的标签由于其向量积均为0，所以互不相关。且在总状态数较少的情况下独热编码更简单高效，另一方面又增加了状态的空间，使算法更加准确。

### 卷积层（C1）

卷积神经网络(Convolutional Neural Network, CNN)：

卷积神经网络是深度学习与计算机视觉中极其重要的网络结构之一，许多及其成功的模型都是基于CNN的，特别是基于ImageNet等的各种图像物体识别算法。卷积神经网络算法相对于传统的图像处理算法有很多优点，其中一个就是其巧妙的避免了复杂耗时的预处理工作（如人工特征的提取），原始图像数据经过CNN可以直接进入下一步的运算，不需要人工的参与。

如前文所说，在图像识别的过程中，图像往往会被看成一个或多个的二维矩阵，本文用到的MINST图像库中，图像库中每一个图片就可以看成是28 \* 28 的二维矩阵，因为它是黑白图片，只有一个颜色通道。如果是彩色图片就会有RGB三个颜色通道，就会被表示为三个二维矩阵。传统的神经网络算法使用的是全链接的方式，这种连接方式是将每一个颜色通道的每一个像素都与下一层的神经元·所连接，这样会使参数的数量指数增长，导致参数的数量巨大，使整个神经网络的训练及其耗时甚至时间数量级巨大，无法训练。而卷积神经网络通过局部链接，权值共享两个主要方法，在几乎不消耗准确率的情况下大幅减少变量数，从而避免这一困难的发生。

（图片）

直观的看，卷积层的参数可以看作是一系列的可以训练和学习的过滤器（卷积核）。在前向学习的过程中，每次我们将图片的每一个小部分通过一个过滤器，点乘后再组成新的二维数据，每个过滤器计算成的二维数据再组成三维数据。我可以理解成每个过滤器只关心一部分图片的特征，再综合图片的各个特征进行下一步的训练。

局部连接：

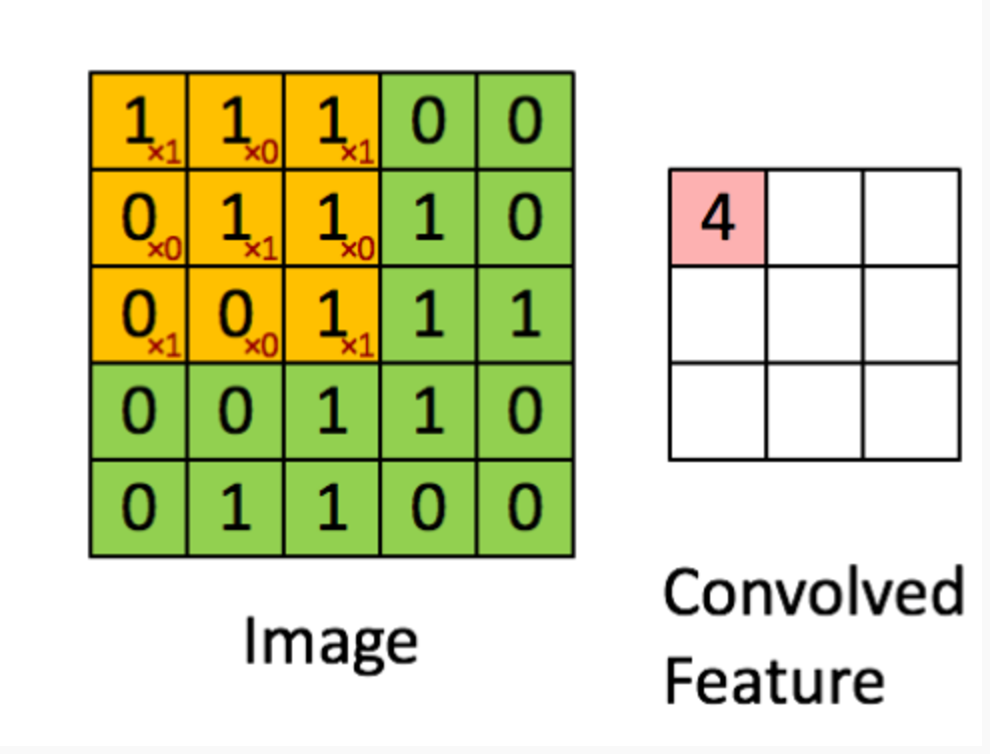
局部连接是卷积神经网络独特的原因之一，上文提到在图片中，使用全连接的神经网络在工程上是不可行的，而局部链接的意思就是卷积神经网络的神经元只会与上层的神经元的局部区域进行连接。可以把上层数据想象成一张画，画的上面有一个滑动的窗口，每次运算时只有窗口内的数据会与下层进行关联。这就引出了卷积神经网络的一个重要的变量：窗口大小，一般窗口的长与宽是相等的，窗口的长\*宽叫做感受野（receptive field），在整个运算中，这个窗口会逐格滑动，一步步近似覆盖图片的每一个小区域。

在本算法中，我们的MINST图片库输入的图片数据大小为[1\*28\*28]，我们设定感受野（窗口）大小为[5\*5]，为了提取不同的特征，我们有6个不同的卷积核。

权值共享：

权值共享是卷积神经网络中很巧妙的处理方方式，它可以使卷积神经网络的训练与识别时的计算复杂度和参数的个数下降非常非常多。在本算法中，我们共有6个卷积核，每个卷积核大小为5 \* 5，所以经过特征图（feature map）的长与宽都为（28-5+1），共有6个特征图，所以每个图片的总变量数位6\*24\*24 = 3456个神经元，每个窗口由于数据的连接，有5\*5=25个权重，所以共有25\*3456= 86400个参数，如果每张图片的计算量都是这么大的话，会变得非常非常慢，也就失去了它的意义。

因为每个卷积的神经元可以看成是一个过滤器，那么我们可以大胆的假设，这个神经元用于连接每个数据窗口的权重都是固定的，不会因为窗口的位置而改变。这意味着，对于每一个卷积神经原来说，不管其对应的数据窗口在图片的哪个位置，其权重都是同一组数。那就代表，我们的卷积层中，我们只需要神经元个数\*卷积核的维度= 6\*6\*5=180个权重。



如果每个神经元在不同位置对应的权重是固定的，那么同一个卷积神经元进行计算卷积积分的整个过程就可以看成是一组固定的权重和不同位置的数据窗口做内积的过程，在数学上刚好对应着『卷积』操作，这也是卷积神经网络名字的来源。

另外，因为每个神经元的权重都是固定的，它们都可以看做一个恒定的过滤器，比如我们的6个卷积神经元作为过滤器后可视化之后的样子如图：



需要说明的一点是，参数共享并不是一个在每个场景都适用的一种策略，在一些特定的场合中，如人脸识别中，由于一般照片中人的面部都集中在画面的中央，所以在人脸识别的过程中，我们不能把照片上的每个位置的窗口数据的作用都视作等同的。在这种情况下，我们希望数据窗口滑过中心区域的时候其权重和其他边缘区域是不同的。对于这种情况我们有一种特殊的层对应这种功能，叫做**局部连接层，**在本算法中局部连接层没有被用到，就暂不赘述。

具体实现：

本算法使用的是批梯度下降法（mini-batch learning）进行训练，即将多个输入数据同时通过神经网络进行计算，计算后将更新的数据进行叠加来更新，这样既会利用python的系统优化提高计算效率，又会避免低效重复的更新，具体将在本章后面有花的部分进行详细描述。

首先初始化经卷积后的多个特征图，由于有每次训练是由多个输入数据组成的批，每批有6个特征图，每个特征图是24（图片宽度-窗口宽度+1）\*24的矩阵，我们先初始化特征图组为批大小\*特征图数\*特征图高\*特征图宽的四维矩阵，然后用每个卷积核分别用窗口划过每张图片，每滑动一次求出相应的积并填入特征图相应的位置中。具体代码如下。

conved\_input = np.zeros((train\_data\_batch.shape[0], num\_of\_fmap, image\_size - conv\_core + 1, image\_size - conv\_core + 1), dtype = np.float)  
**for** i **in** range(0, train\_data\_batch.shape[0]):  
 **for** j **in** range(0, num\_of\_fmap):  
 **for** k **in** range(0, image\_size - conv\_core + 1):  
 **for** l **in** range(0, image\_size - conv\_core + 1):  
 conved\_input[i][j][k][l] = sum(sum(two\_d\_input[i][k:k + conv\_core, l:l + conv\_core] \* (in\_to\_conv\_weights[j]. reshape(conv\_core, conv\_core))))

由于本程序是基于CPU，所以每个求积需要单独运算，本算法经过GPU加速后可以根据GPU的核心数（如NVIDIA GeForce GTX1050Ti有640个核心）同时运算，可大幅提高速度。

### 池化层

图？

### 一般来说，在卷积神经网络中，池化层是在连续的卷积层中间或者卷积到全连接层中间的层。它的作用也非常简单，就是通过抽样的方法逐步地减少和压缩数据和参数的量，在一定程度上也会减小过拟合的现象，即过度的符合训练数据而使容错性降低。池化层做的操作也非常简单，就是将原数据上的每个小区域压缩成一个值(小区域的区域最大值（MAX）或者平均值（AVERAGE）)，最常见的池化层的设定是，将原数据切成由2\*2的小块组成的格式，在每块里面取最大值作为输出，这样我们就自然而然减少了75%的数据量，抽样的方式根据分数据特点的不同而不同。

### 在最大值和平均值的池化方式外，我们也可以设定其他的抽样方式，如L2范数池化。

本算法中对池化层和它的操作的直观理解的示意图为：



如图为本算法中池化层的一个直观的示意，即对上一层（卷积层）处理后得出的6\*24\*24的数据（一般来说，特征图的数量被称为厚度）的每一个切片（1\*24\*24的部分）做了一个下采样。下图为次采样操作的实际操作。



具体实现：

在本算法中，通过上文我们知道，由上层卷积神经网络计算出了维度为批大小\*特征图数\*特征图高\*特征图宽的四维矩阵，我们对每个特征度进行取样单位为2\*2，的最大值取样，所以取样后的特征图的维度为批大小\*特征图数\*（特征图高／2）\*（特征图宽／2），对于每个取样后的特征图的位置，其值为对应的四个特征图的值的最大值。为了方便后面更新权制，需要把最大值所在位置的坐标也同时记录下来，具体代码如下：

conved\_input\_maxloc = np.zeros( (train\_data\_batch.shape[0], num\_of\_fmap, image\_size - conv\_core + 1, image\_size - conv\_core + 1),dtype=np.float)

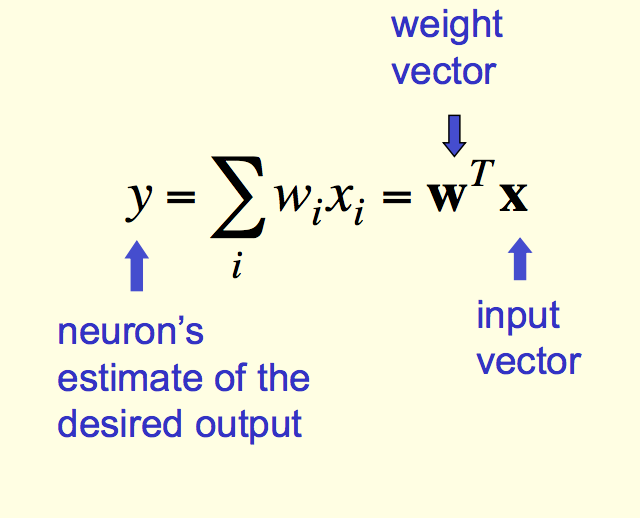
sampled\_input = np.zeros((train\_data\_batch.shape[0], num\_of\_fmap, int((image\_size - conv\_core + 1) / 2), int((image\_size - conv\_core + 1) / 2)), dtype=np.float)  
**for** i **in** range(0, train\_data\_batch.shape[0]):  
 **for** j **in** range(0, num\_of\_fmap):  
 **for** k **in** range(0, int(conved\_input.shape[2] / 2)):  
 **for** l **in** range(0, int(conved\_input.shape[3] / 2)):  
 sampled\_input[i][j][k][l] = np.amax(conved\_input[i][j][2 \* k:2 \* k + 2, 2 \* l:2 \* l + 2])  
 agmx = np.argmax(conved\_input[i][j][2 \* k:2 \* k + 2, 2 \* l:2 \* l + 2])  
 conved\_input\_maxloc[i][j][2 \* k + int(agmx / 2)][2 \* l + agmx % 2] = 1

### 全连接层

全连接神经网络是神经网络中最基本，最标准的形式。全连接的意思是，神经网络中一层的所有神经元都和上一层的所有神经元之间进行关联，相对于卷积神经网络，全连接神经网络的矩阵运算比较和直接。现代的很多卷积神经网络结构，其末层或末几层都会采用全连接去学习更多的信息。要理解全连接层的工作原理，我们必须了解组成全连接层与所有神经网络的基本单元——神经元的工作原理。



在神经网络中有很多种类的神经元，在全连接神经网络中最常见的神经元叫做现行神经元，又叫做线性过滤器，线性神经元的有一系列的输入，它的参数是这些输入的权值，输出是它输入值的加权和。



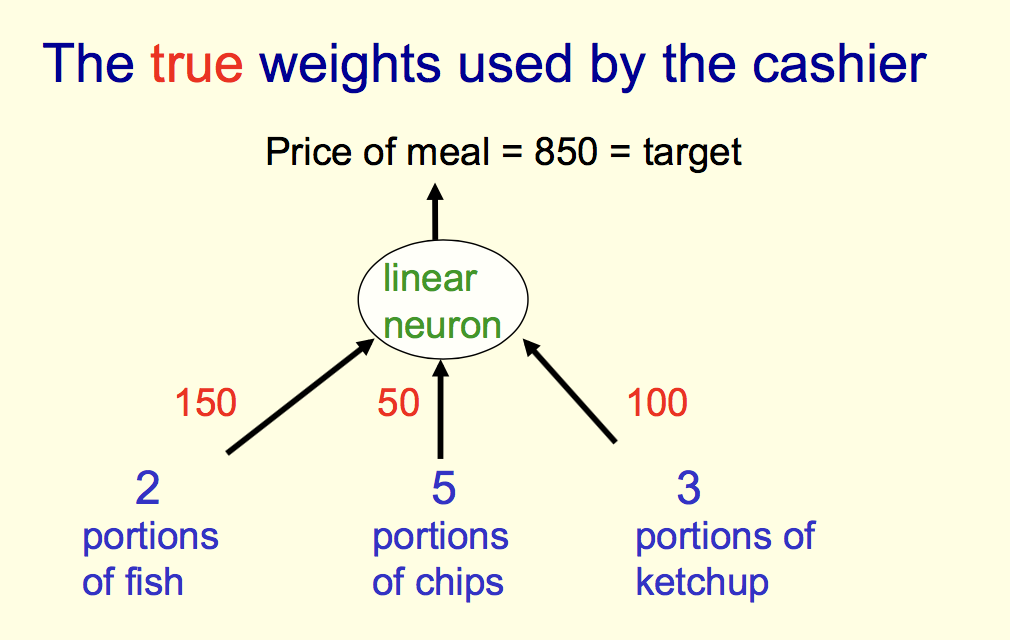
神经元的偏差方程是用来计算神经元输出值与正确值差别的方程，对于不同的情况，有不同的计算偏差的方式，在数字预测方面经常会使用方差模型，在坐标方面有欧式距离和曼哈顿距离，在文字处理方面经常使用交叉熵作为偏差方程。神经元的目标是通过更新自身的参数（对每个输入的权值）来尽量减少输出的偏差，使自身更佳准确，在这里，偏差是所有残差的平方和。

一个神经元的例子：

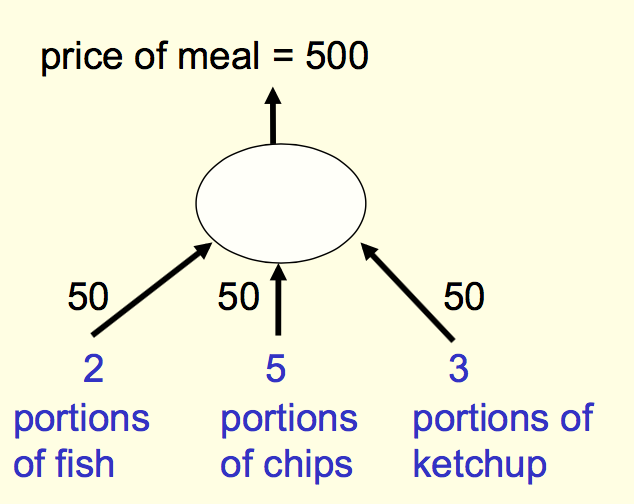
假设你每天在食堂吃午餐，每天的午餐由不同数量的鱼，菜，米饭组成，每次点完餐阿姨只会告诉你总价格，几天之后，你就可以推算出每种食物的单价了。在这里假设你就是一个线性神经元：

每个种类的价格组成了线性神经元的权值：

开始的时候我们会随机猜每种菜品的价格，然后每过一天，通过我们点的每种菜的数量和阿姨告诉我们的总价格，一点点更新我们的猜想，使其更加接近正确的值。



我们首先预测每种食物的价格都为50元（这里是初始值，可以为任何非零数，如果是零会导致更新时的变量为0，无法更新，在卷积神经网络中有时候会用到这个方法保持权值的同步）



在这里输入值为:

线性神经元权值

残差

根据学习规则（delta-rule）权值的变化量 其中为学习率，即学习的速度。

假定学习率，则鱼，菜，饭的变化分别为

更新后的权值为 进一步接近正确值，当有新的数据，循环的次数越来越多之后，预测的价格会越来越接近于真实的价值。这个方法的优化后的标准化方法叫做梯度下降，用偏导的方法找到使误差下降最快的变化量。

梯度下降：

梯度下降又被称为最速下降法，是一种迭代的最优化算法，函数在一个位置的梯度可以表示为这个函数在这一点的时候它最“陡峭”的方向，那么如果一个点向它对应方向的梯度增加很小的数量，那么这个更新后的点的函数值一定会下降。



而梯度下降法正是利用了这样特点，把每个神经元的参数当作这个上点的“坐标”而这个函数就是神经元权值与整个神经网络的偏差的关系，那么利用梯度下降的方法，经过很多很多次的迭代，不断的更新神经元的权值，整个申请网络的偏差就会不断的下降，最终达到一个最小值。

全连接神经网络：

全连接神经网络又多层神经元组成，第n层的神经元的输入与第n-1层的神经元的输出进行相连，同时第n层神经元的输出与第n+1层神经元的输入相连接，这样一层一层传递数据，最终传递出结果，再通过输出结果一层一层的向前更新变量（back-propagation）。通过多层神经网络的数据传递与权值更新，其能通过训练找到数据与其标签的隐藏含义，而相对于单层神经元组成的网络，其能更好的识别更加复杂和隐含的模型。多层神经网络在结构上分为输入层（input layer），隐藏层（hidden layer），和输出层（output layer），其中隐藏层可以有多个。

图

在多层神经网络中，由于是全连接的结构，第n+1层与第n层之间的连接数为他们的神经元数的积，在这里表示位于第n层神经元的权值，表示从上一层第i个神经元连向下一层第j个神经元的权值。

图

在计算的过程中，第n层的第j个神经元的输出值等于第n-1层每个神经元的输出值与其对第n层第i个神经元加权总和

全连接神经网络的计算极其相似于矩阵的乘法，现假设每个批次只有一组输入数据，输入数据为一横行的数据：

设矩阵W[i][j]表示输入数据到第一层隐藏层的权值，纵坐标表示输入神经元i，横坐标表示输出神经元j，W[i][j]表示

将输入矩阵与权制矩阵点乘，通过点乘法则得到矩阵：

正好为隐藏层的输出值，这个数组可以在与下一层的权值进行点乘，如此就可以依次迭代快速得出所有层的值。

在批梯度下降法中，可以将一个批次中每行数据并成一个批次大小\*数据维度的矩阵，这样可以通过矩阵与矩阵的乘法快速高效的点乘出下一层的值：

具体实现：

在本算法中，由上层卷积神经网络经池化后的数据维度为批大小 \* 特征图数 \*（特征图高／2）\*（特征图宽／2），在这里为了更好的利用矩阵的特性进行全连接计算，将池化后的数据进行处理，将维度变化为二维，纵向为批大小，将每批的数据转换为1行，这样输入到全连接神经网络的数据就会变成批大小 \* （特征图数 \*（特征图高／2）2，综合速度与准确性的考虑，本算法中我们设定隐藏层大小为200 个线性神经元，输入数据经过与权值的矩阵点乘后，为了使训练更加高效，又将点乘后的数据加入了可训练的偏差值后得到了隐藏层。由于输出结构为之前独热编码后的0-9的数字，所以有10个输出的神经元。用同样的方法由隐藏层计算出输出层后，由于本程序的宗旨是预测数字，所以比较合适的偏差是计算机对于其认为图像为每个数字概率。在这里使用将输出层映射到sigmoid函数的方法，对于sigmoid函数的作用，将在优化的部分详尽解释。

图

具体代码：

oned\_fnn\_in = sampled\_input.reshape(sampled\_input.shape[0], sampled\_input.shape[1] \* sampled\_input.shape[2] \* sampled\_input.shape[3])

hid\_state = np.dot(train\_batch, in\_to\_hid\_weights)  
hid\_state = hid\_state + hid\_bias  
out\_state = np.dot(hid\_state, hid\_to\_out\_weights)  
out\_state = out\_state + out\_bias  
out\_state = 1 / (1 + np.exp(-out\_state))

### 输出层

输出层的作用很简单，将上层输出的神经网络求出的每个数概率中，选出最大的一项输出结果。

为了测试，也可以输出其他的结果，比如概率最高的N个数或保证一定概率的置信数组。

具体代码：

prediction = np.argmax(out\_state, axis=1)

### 权值更新

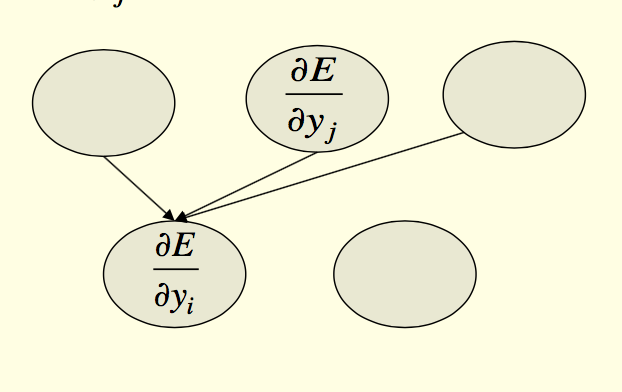
前文提到，神经元是通过梯度下降的方法来使自身变得准确的，而神经网络是通过其当中每一个神经元的梯度下降来提高整个神经网络的准确性的。相对于一个一个为神经元求其对整个神经网络残差的偏导数，利用偏导数的链式法则（）神经网络利用反向传播的方法来按层批量更新权值。

反向传播：

反向传播的基本思想是梯度下降，对于复杂的神经网络，我们无法通过解方程的方式得到每个权值的最佳值，但是我们可以通过求梯度的方式来得到每个神经元的每个权值在变化量足够小的时候其相对于最佳值的方向，对于复杂的神经网络，由于上层神经元的输出会与多个下层神经元进行连接，所以对于这样的神经元来说，其变化量其实是其所有连接的神经元对结果的影响和的加权值。相对于单独计算，反向传播大大提高了运算效率。

全连接神经网络的反向传播：

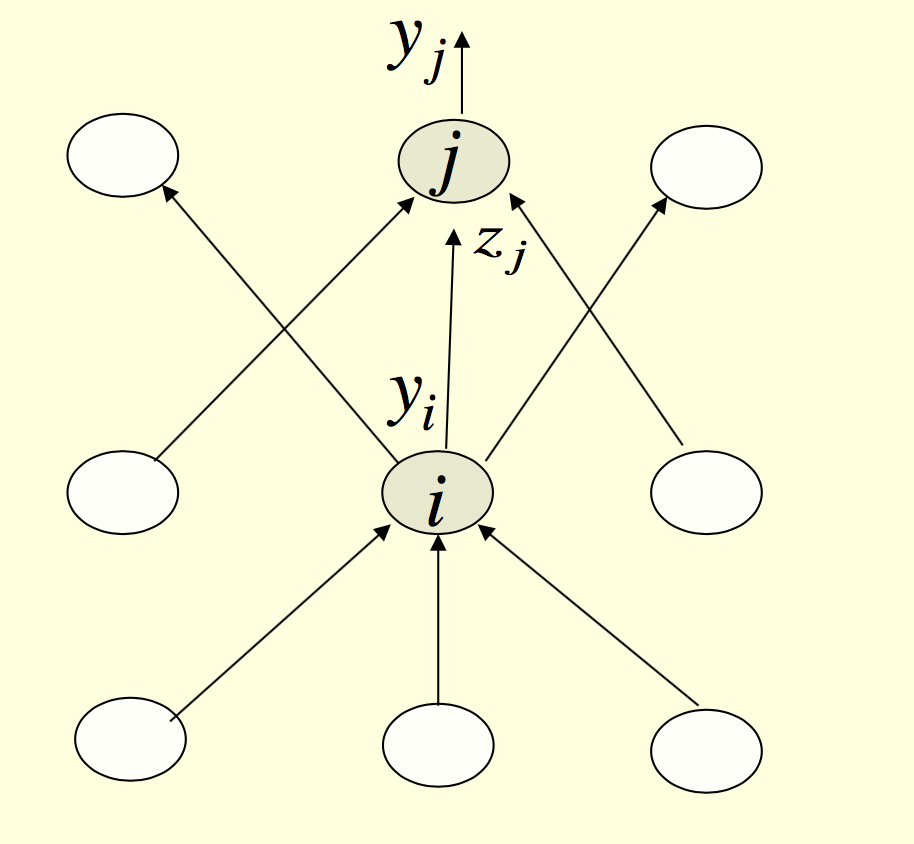
之所以叫反向传播，是因为这个方法是按照从输入到输出的顺序一层一层向前推进的，这样巧妙的利用了链式法则的优势。



在这里神经元j是输出层的神经元，由图可知，上一层神经元的偏导数实际上是由下一层与其连接的神经元的偏导数叠加而成。

我们假设此网络的偏差方程为方差：

那么输出层神经元j相对于偏差E的偏导数



对于输出层，上文提到，输出层神经元经常会用到像sigmoid或softmax的函数进行优化，所以这里神经元j的输入可能并不与输出成线性相关的关系：

在这里举例使用本算法的sigmoid神经元作为输出神经元：

则下一层的神经元i，因为对于下一层神经元j，神经元相当于对其做了权值Wij的‘贡献’，因此其相对于偏差E的偏导数可以理解为为上一层所有神经元的输入值偏导数的加权和：

对于每层神经元来说，以权值的角度看，可以理解成下层神经元是输入其中的权值的“以输入值为权值的加权和”那么权值的偏导数实际上就是其指向的神经元的偏导数于其对应输入值的积：

按此方法可以依次向前推进，依次求出全连接神经网络中输入层到隐藏层权值的偏导值。如果全连接层前面有其他的神经网络层，也需要求出其输入层的偏导值以进行下一步运算。

卷积神经网络的反向传播：

卷积神经网络的反向传播思想于全连接神经网络大同小异，其宗旨都是找到神经元或权值当把它们当成下一层输入的时候的权值，再用这个权值与其连接下一层的偏导数进行加权乘积。

图：

在本算法卷积神经网络中，由于在卷积之后有最大值池化的优化，所以在求偏导数时需要将池化中筛选掉的输入值同时筛选掉。筛选掉后对于每个卷积核的权值，找到与其进行加权乘积后的输入值，再进行点乘后得出其偏微分。

实现：

在本算法中，输出层为sigmoid神经元，其维度为批大小\*10，结构为：

根据前面推导的sigmoid神经元的偏导公式，输出层的偏导矩阵为：

通过输出层神经元偏导矩阵与隐藏层输出值矩阵的转置进行点乘可以得到隐藏层到输出层权值的偏导，其维度为隐层神经元个数\*输出神经元个数，与隐层神经元到输出神经元的权值矩阵相同：

由于偏移量没有权值，所以起偏导就等于其连接的神经元的偏导和：

通过输出层偏导矩阵与隐藏层到输出层权值转置进行点乘，得到隐藏层神经元的偏导矩阵，矩阵的维度为批大小\*隐层神经元个数，与隐层神经元矩阵相同：

通过同样的方法依次求出输入层到输出层权值偏导矩阵，输入层偏导矩阵，隐藏层偏移量偏导：

再利用全连接神经网络输入层的偏导来求卷积核的权值的偏导数：

本算法中，先利用之前记录下来的最大值的坐标与卷积神经网络的输入数据，合成一个维度与输入数据相同，只保留最大值池化（将池化没有选中的设为零）的数组。再直接用卷积核上的想要求出卷积的那个参数所有“划过”的点乘上其对应的输入向量，最后加和：

再将已经求出的所有的偏微分乘上学习率，用其对应的权值去减去求出的变化量，久完成了权值的更新。

具体代码：

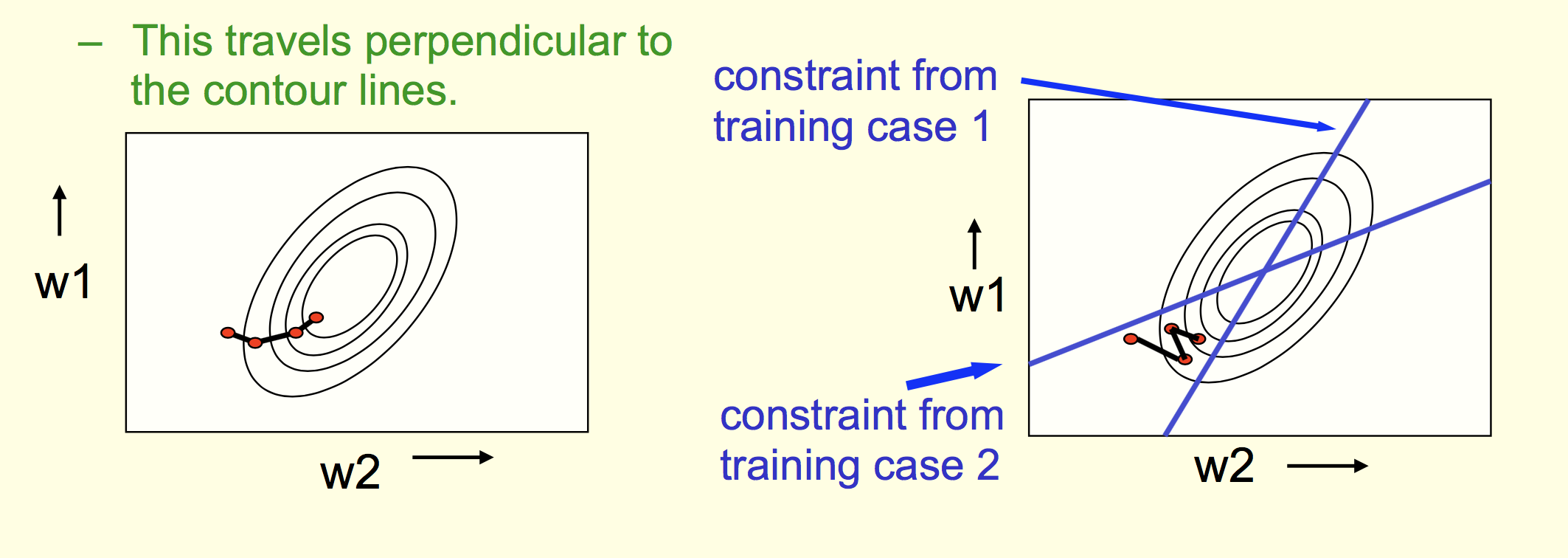
*# bprop:err*err = out\_state - train\_label\_batch  
CE = (err \* err / 2).sum(axis=1)  
d\_Out = out\_state \* (1 - out\_state) \* err  
d\_hid\_to\_out = np.dot(hid\_state.transpose(), d\_Out)  
d\_hid = np.dot(d\_Out, hid\_to\_out\_weights.transpose())  
d\_in\_to\_hid = np.dot(train\_batch.transpose(), d\_hid)  
d\_in = np.dot(d\_hid, in\_to\_hid\_weights.transpose())  
  
*# update weights*del\_hid\_to\_out = -1 \* alpha \* d\_hid\_to\_out / batch\_size  
del\_in\_to\_hid = -1 \* alpha \* d\_in\_to\_hid / batch\_size  
hid\_to\_out\_weights += del\_hid\_to\_out  
in\_to\_hid\_weights += del\_in\_to\_hid  
  
hid\_bias += -1 \* alpha \* d\_hid  
out\_bias += -1 \* alpha \* d\_Out

d\_in = d\_in.reshape(d\_in.shape[0], num\_of\_fmap, int((28 - conv\_core + 1) / 2),  
 int((28 - conv\_core + 1) / 2))  
expd\_d\_in = np.zeros((d\_in.shape[0], d\_in.shape[1], d\_in.shape[2] \* 2, d\_in.shape[3] \* 2), dtype=np.float)  
*# print(d\_in.shape)*kr = [[1, 1], [1, 1]]  
**for** i **in** range(0, train\_data\_batch.shape[0]):  
 **for** j **in** range(0, num\_of\_fmap):  
 expd\_d\_in[i][j] = np.kron(d\_in[i][j], kr)  
*# print(expd\_d\_in[0][0])*expd\_d\_in = expd\_d\_in \* conved\_input\_maxloc  
*# print(expd\_d\_in[0][5])  
# print(train\_data\_batch)*d\_cov\_w = np.zeros((d\_in.shape[1], conv\_core, conv\_core), dtype=np.float)  
**for** i **in** range(0, train\_data\_batch.shape[0]):  
 **for** j **in** range(0, num\_of\_fmap):  
 **for** k **in** range(0, conv\_core):  
 **for** l **in** range(0, conv\_core):  
 d\_cov\_w[j][k][l] += sum(sum(  
 expd\_d\_in[i][j] \* two\_d\_input[i][k:k + expd\_d\_in.shape[2], l:l + expd\_d\_in.shape[3]]))in\_to\_conv\_weights += -alpha \* (d\_cov\_w.reshape(d\_cov\_w.shape[0], d\_cov\_w.shape[1] \* d\_cov\_w.shape[2]))

# 实验分析与优化

在本算法中，对算法运算时间，准确率，效率，容错性等特性有影响的参数有很多，如卷积核大小，池化层的池化程度，学习率，有无偏移量，隐藏层的神经元数量，隐藏层的层数等。通过调整各个参数的数量，来对算法进行优化，下面通过实验与分析了解各个参数的作用与在本算法中的最佳选择：

## 批的大小（batch size）



在线上学习（每次只学习一个图像）中，由于对于复杂的神经网络，其残差函数比较负载，在梯度下降时，可能出现每次下降的方向不一样，出现zigzag的现象，

zigzag向量图：

通过批学习的方法，用同一组权值同时训练多组输入数据，将每组数据的变化量加和后更新权值，用这种方法可以通过抵消掉其他方向的重合部分，从而提高更新权值的效率。以下是在其它参数固定的情况下，批大小分别为1，10，100，1000时训练准确率与训练进度的图像和训练准确率与训练时间的图像：



由图像我们可以看出，在合理的批的大小范围内，在训练进度相同时，神经网络的准确率几乎是相同的，此时批的大小对训练精度的影响几乎可以忽略不计；但是在训练时间上可以出，在合理的批大小范围内，随着批大小的增大，训练的速度会明显提升。儿训练后的最终精度并不会受到明显的影响。但是当批的大小过大时（batch size=1000），由于每次更新的权值变化量积累过多，会造成过量更新，使权值无法接近最优点，造成训练失败，对于某些情况，可以用减少学习率的方法来解决这个问题。

## 训练次数（epoch）

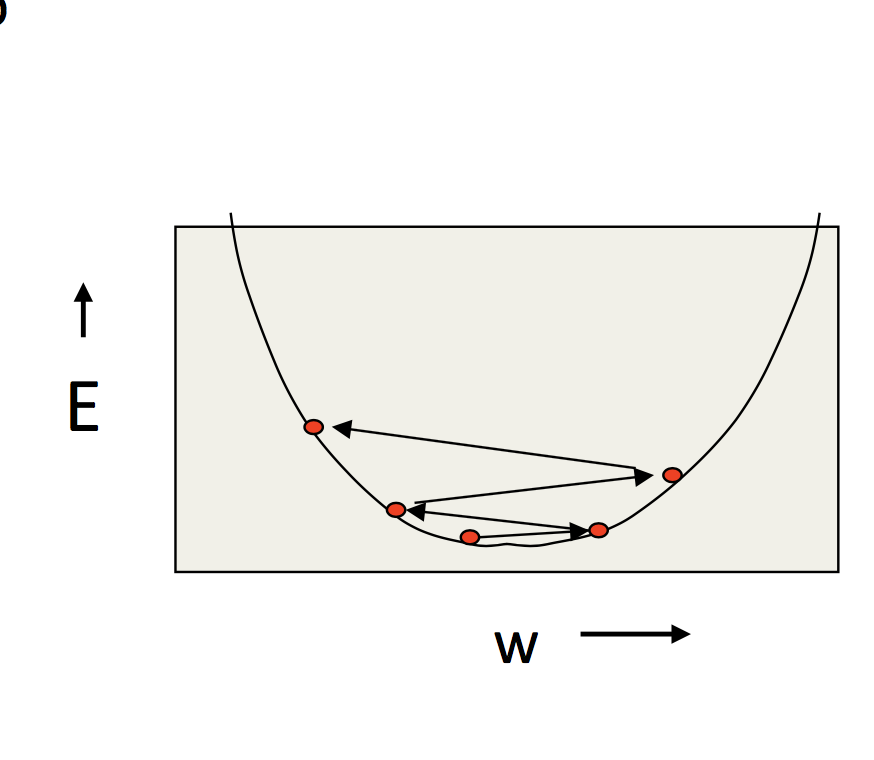
在机器学习的过程中，由于数据量有限且步长（学习率）较小，特别是当批尺寸比较大的时候，由于权值更新的次数有限，权值在一次训练过并没有达到最优解，这时候可以使用多次训练的方法使其充分学习，批尺寸越大的时候其达到较高精度所需要的训练次数就越多。在训练达到算法精度的极限后，算法的预测准确性不会再因为扩大训练次数而提高：



由图像可以看出，当批大小为1和10时，神经网络的准确率在第3次训练后不再增加，当批大小为200时，神经网络的准确率在第5次训练后不再增加。

## 学习率（learning rate）

因为每次权值的更新变化量都是其偏导与学习率的乘积，所以学习率在一定程度上直接决定着神经网络学习的速度。对于过小的学习率，神经网络对训练数据会不能有效利用，神经网络的学习速度会很慢，对于过大的学习率，由于每次更新的权值过大，会出现“跳过”最优解的情况，而离最优解更远之后会造成其所在权值偏导增大，求出的权值变化量更大，进一步远离最优解，最后造成学习失败。



图：我们使用如下几组参数的学习进行对比：

## 神经元数量：

## 神经元层数：

## 有无卷积

## 卷机核大小

## 池化层数：

## 偏移量：

## 标准化：

优化与测评：

优化：batch normalization bias