# 阅读笔记：

深度神经网络是一种模块化的模型，它由一些列作用在数据块之上的内部连接层组合而成。根据Caffe[官方文档](http://caffe.berkeleyvision.org/tutorial/net_layer_blob.html" \t "http://blog.csdn.net/t0903/article/details/_blank)介绍，Caffe大致可以分为三层结构blob，layer，net。Caffe基于自己的模型架构，通过逐层定义的方式定义一个网络，网络从数据输入层到损失层自下而上地定义整个模型 。Caffe 使用 blobs 结构来存储、交换和处理网络中正向和反向迭代时的数据和导数信息：blob 是 Caffe 的标准数组结构，它提供了一个统一的内存接口。 Layer 是 Caffe 模型和计算的基本单元，Net 是一系列 layers 和其连接的集合。 Blob是标准的数组结构，它提供了统一的内存接口， 详细描述了信息是如何在 layer 和 netnet 中存储和交换的。

# 721[Caffe学习笔记2-Caffe的三级结构(Blobs,Layers,Nets)](http://blog.csdn.net/t0903/article/details/48266169)

## 1.Caffe总体架构

Caffe框架主要有五个组件：**Blob**，**Solver**，**Net**，**Layer**，**Proto**，其结构图如下图1所示。**Solver**负责深度网络的训练，每个Solver中包含一个训练网络对象和一个**[测试](http://lib.csdn.net/base/softwaretest" \t "http://blog.csdn.net/wspba/article/details/_blank" \o "软件测试知识库)**网络对象。每个网络则由若干个**Layer**构成。每个Layer的输入和输出Feature map表示为Input Blob和Output Blob。**Blob**是Caffe实际存储数据的结构，是一个不定维的矩阵，在Caffe中一般用来表示一个拉直的四维矩阵，四个维度分别对应Batch **Size（N）**，Feature Map的通道数**Channel（C）**,Feature Map高度**Height(H)**和宽度**Width(W)**。**Proto**则基于Google的Protobuf开源项目，是一种类似XML的数据交换格式，用户只需要按格式定义对象的数据成员，可以在多种语言中实现对象的序列化与反序列化，在Caffe中用于网络模型的结构定义、存储和读取。

## 模型的格式

模型定义在.prototxt文件中，训练好的模型在model目录下.binaryproto格式的文件中。模型的格式由caffe.proto定义。采用Google Protocol Buffer可以节省空间还有它对C++和Pyhton的支持也很好。

# caffe的matlab和python接口安装

### 无法打开包括文件

使用微软的caffe版本([https://github.com/Microsoft/caffe](https://github.com/Microsoft/caffe" \t "http://blog.csdn.net/hereiskxm/article/details/_blank))时遇到了"gpu/mxGPUArray.h" Not Found问题，是因为的matlab版本太新了，修改CommonSettings.props配置如下：

<IncludePath>$(MatlabDir)\extern\include;$(MatlabDir)\toolbox\distcomp\gpu\extern\include;$(IncludePath)</IncludePath>  <!-- 增加了一个include路径 -->

具体排查过程，全局查找mxGPUArray.h，发现在$(MatlabDir)\toolbox\distcomp\gpu\extern\include路径下，遂在配置中加上。估计是matlab新版本中gpu文件夹已从\extern\include独立出来了。

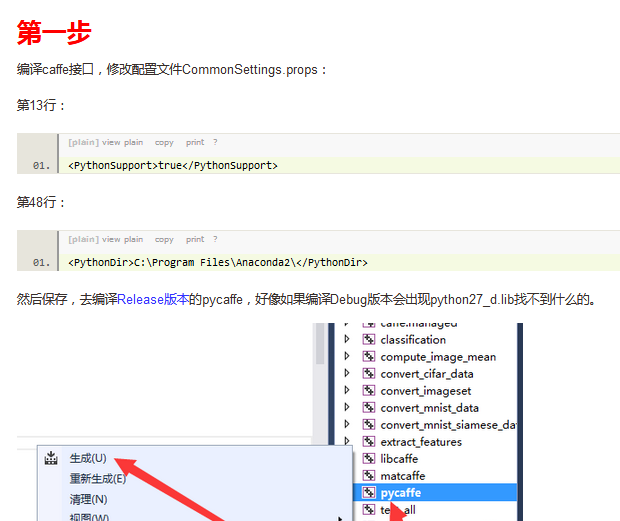
**Python：**

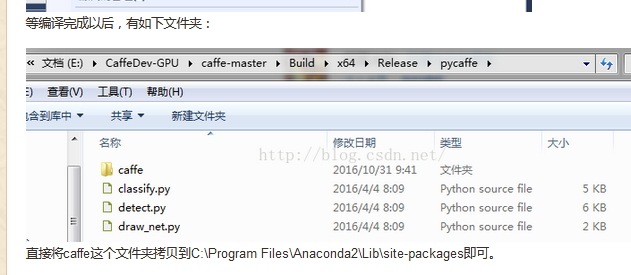
:无法打开包括文件 "pyconfig" no such file or directory

Python的路径换成绝对路径

<IncludePath>$(PythonDir)\include;D:\AnZhuangbao\p\_anaconda\include;$(IncludePath)</IncludePath>

### 一：python的importcaffe





放在python安装目录下

2、[No module named google.protobuf.internal](http://blog.csdn.net/wuzuyu365/article/details/52431062)

Pip install protobuf

## Matlab接口参考：

<http://blog.csdn.net/zb1165048017/article/details/51702686>

重启后生效,电脑重启

## Python接口：

1.2 编译生成pycaffe   
编译解决方案，注意需要在【release】模式下，因为在debug下找不到 python对应的调试版本。   
在[caffe\_root]\Build\x64\Release 目录下会生成pycaffe目录。我一般会把内部的caffe文件夹直接拷贝到python包管理目录site-packages中。

# 719无法打开包括文件caffe

# 卷积神经网络(cnn)

自从AlexNet在2012年的成功以来，CNN已有各种改进。在本节中，我们**从卷积层，池层，激活函数，损耗函数，正则化和优化六**个方面描述CNN的主要改进。

而卷积神经网络中conv+relu(早期为sigmoid)+pooling(以下称三剑客)的组合，不仅可以**替代手工设计特征算子的繁琐**，而且**局部感受野+权值共享的设计思想也能避免全连接网络中的种种弊端。**此时人们将三剑客的组合视为**特征提取**的过程，如果按照早期人们特征提取+分类的设计思路，那么分类使用全连接的设计方式，就可以刚好实现了一个end-to-end的架构，也即早起卷积神经网络的原型。

是近年发展起来的一种高效识别方法，在各种图像处理任务中大放异彩。

卷积神经网络由卷积层、激活函数、池化层、全连接层组成

1、卷积层：特征提取卷积层（局部感知、权值共享、多卷积核）

2、池化层：一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征（降维，特征选择）

3、全连接层：连接所有的特征，将输出值送给分类器（如softmax分类器）。（全连接+softmax）

### 全连接层的作用：

作者：魏秀参 知乎

 全连接层（FC）在整个卷积神经网络中**起到“分类器”的作用**。如果说卷积层、池化层和激活函数层等操作是将原始数据映射到隐层特征空间的话，全连接层则起到将学到的“分布式特征表示”映射到样本标记空间的作用。在实际使用中，全连接层可由卷积操作实现：对前层是全连接的全连接层可以转化为卷积核为1x1的卷积；而前层是卷积层的全连接层可以转化为卷积核为hxw的全局卷积，h和w分别为前层卷积结果的高和宽（***注1***）。

 目前由于全连接层参数冗余（仅全连接层参数就可占整个网络参数80%左右），近期一些性能优异的网络模型如ResNet和GoogLeNet等均用**全局平均池化**（GAP）取代FC来融合学到的深度特征，最后仍用**softmax等损失函数作为网络目标函数**来指导学习过程。需要指出的是，用GAP替代FC的网络通常有较好的预测性能。

 在FC越来越不被看好的当下，我们近期的研究发现，FC可在模型表示能力迁移过程中**充当“防火墙”的作用**。具体来讲，假设在ImageNet上预训练得到的模型为IMG_256 ，则ImageNet可视为源域（迁移学习中的source domain）。**微调（fine tuning）是深度学习领域最常用的迁移学习技术。**针对微调，若目标域（target domain）中的图像与源域中图像差异巨大（如相比ImageNet，目标域图像不是物体为中心的图像，而是风景照，见下图），不含FC的网络微调后的结果要差于含FC的网络。因此FC可视作模型表示能力的“防火墙”，特别是在源域与目标域差异较大的情况下，**FC可保持较大的模型capacity从而保证模型表示能力的迁移。（**冗余的参数并不一无是处。）

### 卷积层的设计原则

1、局部感知野：图像具有空间相关性。局部的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息

1000×1000 、1000000×1000000=10^12、假如每个神经元只和10×10个像素值相连，那么权值数据为1000000×100个参数，减少为原来的千分之一。

其实这样的话参数仍然过多，那么就启动第二级神器，即权值共享。在上面的局部连接中，每个神经元都对应100个参数，一共1000000个神经元，如果这1000000个神经元的100个参数都是相等的，那么参数数目就变为100。我们可以这100个参数（也就是卷积操作）看成是提取特征的方式，该方式与位置无关。

2、权值共享：图像的一部分的统计特性与其他部分是一样的。这也意味着我们在这一部分学习的特征也能用在另一部分上，所以对于这个图像上的所有位置，我们都能使用同样的学习特征。

例子：更直观一些，当从一个大尺寸图像中随机选取一小块，比如说 8×8 作为样本，并且从这个小块样本中学习到了一些特征，这时我们可以把从这个 8×8 样本中学习到的特征作为探测器，应用到这个图像的任意地方中去。特别是，我们可以用从 8×8 样本中所学习到的特征跟原本的大尺寸图像作卷积，从而对这个大尺寸图像上的任一位置获得一个不同特征的激活值。

3、多卷积核：如果只有一个卷积核。特征提取是不充分的，我们可以添加多个卷积核，比如32个卷积核，可以学习32种特征。在有多个卷积核时

上图右，不同颜色表明不同的卷积核。每个卷积核都会将图像生成为另一幅图像。比如两个卷积核就可以将生成两幅图像，这两幅图像可以看做是一张图像的不同的通道。

2、池化层

在通过卷积获得了特征 (features) 之后，下一步我们希望利用这些特征去做分类。理论上讲，人们可以用所有提取得到的特征去训练分类器，例如 softmax 分类器，

但这样做面临计算量的挑战。如果我们把所有提取到的特征用于训练分类器，那么意味着要学习一个拥有超过数百万特征输入的分类器，这十分不便，并且容易出现过拟合。

池化层：一方面使特征图变小，简化网络计算复杂度；一方面进行特征压缩，提取主要特征

为了描述大的图像，可以对不同位置的特征进行聚合统计，例如，人们可以计算图像一个区域上的某个特定特征的平均值 (或最大值)。这些概要统计特征不仅具有低得多的维度 ，可以减少计算量，同时还不容易过拟合

### 激活函数Relu;

## 梯度消失

Sigmoid 函数的导数最大值为 0.25，这意味着梯度每传播一层至少变为原来的 1/4，网络越深底层求解到的梯度越小，这样的情况下网络参数会因为获得的梯度值较小而无法得到有效的更新甚至会不更新，最终会导致网络始终无法收敛，这种现象称之为梯度消失。一般来说，网络深度在 5层之内就会发生**梯度消失**的现象

另外，Sigmoid 函数的输出始终是大于 0 的，这会导致位于其后的网络层的输入是**非零均值**的信号。假设后层的输入都是正的，那么对网络参数求得的局部梯度也都是正的，会导致**收敛缓慢。**这个问题可以通过批训练的方式解决，因为一批数据里求得的梯度总是有正有负的

1激活函数：Relu; 只对前两个全连接层进行dropout

sigmoid，但实际梯度下降中，sigmoid容易饱和、造成终止梯度传递，且没有0中心化。咋办呢，可以尝试另外一个激活函数：ReLU，其图形表示如下

　常用的非线性激活函数有sigmoid、tanh、relu等等，前两者sigmoid/tanh比较常见于全链接层，后者relu常见于卷积层。**ReLU的优点是收敛快，求梯度简单**

2

上图中CNN要做的事情是：给定一张图片，是车还是马未知，是什么车也未知，现在需要模型判断这张图片里具体是一个什么东西，总之输出一个结果：如果是车 那是什么车

    所以

* 最左边是数据输入层，对数据做一些处理，比如去均值（把输入数据各个维度都中心化为0，避免数据过多偏差，影响训练效果）、归一化（把所有的数据都归一到同样的范围）、PCA/白化等等。CNN只对训练集做“去均值”这一步。

    中间是

* CONV：卷积计算层，线性乘积 求和。
* RELU：激励层，上文2.2节中有提到：ReLU是激活函数的一种。
* POOL：池化层，简言之，即取区域平均或最大。

    最右边是

* FC：全连接层

    这几个部分中，卷积计算层是CNN的核心，下文将重点阐述。

而未知图案的局部和标准X图案的局部一个一个比对时的计算过程，便是卷积操作。卷积计算结果为1表示匹配，否则不匹配。

### 总结零碎

1深度学习dp，cnn是深度学习的一种算法，cnn有很懂模型

2常用的**非线性激活函数**有sigmoid、tanh、relu等等，前两者sigmoid/tanh比较常见于全连接层，后者relu常见于卷积层。这里先简要介绍下最基础的sigmoid函数（btw，

### 区别CNN、RNN，dnn

CNN 专门解决图像问题的，可用把它看作特征提取层，放在输入层上，最后用MLP 做分类。

RNN 专门解决时间序列问题的，用来提取时间序列信息，放在特征提取层

DNN 说白了就是 多层网络，只是用了很多技巧，让它能够 deep 。

### 4.4 GIF动态卷积图

    在CNN中，滤波器filter（带着一组固定权重的神经元）对局部输入数据进行卷积计算。每计算完一个数据窗口内的局部数据后，数据窗口不断平移滑动，直到计算完所有数据。这个过程中，有这么几个参数：   
　　a. 深度depth：神经元个数，决定输出的depth厚度。同时代表滤波器个数。  
　　b. 步长stride：决定滑动多少步可以到边缘。

　　c. 填充值zero-padding：在外围边缘补充若干圈0，方便从初始位置以步长为单位可以刚好滑倒末尾位置，通俗地讲就是为了总长能被步长整除。

平铺卷积，转置卷积：代码？？Dilated CNN

### 池化

**卷积神经网络中，除了卷积层，还有一种叫池化的操作**。池化操作在统计上的概念更明确，就是一个对一个小区域内求平均值或者求最大值的统计操作

Stochastic Pooling。 Spectral Pooling。Spatial Pyramid Pooling。Multi-scale Orderless Pooling

## 重叠池化

金字塔池化

### 卷积核/卷积

：卷积是图像处理常用的方法,给定输入图像,在输出图像中每一个像素是输入图像中一个小区域中像素的加权平均,其中**权值由一个函数定义,这个函数称为卷积核,**

1什么是**卷积**？卷积就是两个函数之间的相互关系，然后得出一个新的值，他是在连续空间做积分计算，然后在离散空间内求和的过程。实际上在计算机视觉里面，可以把卷积当做一个抽象的过程，就是把小区域内的信息统计抽象出来

2对图像（不同的数据窗口数据）和滤波矩阵（一组固定的权重：因为每个神经元的多个权重固定，所以又可以看做一个恒定的滤波器filter）做**内积**（逐个元素相乘再求和）的操作就是所谓的『卷积』操作，也是卷积神经网络的名字来源。

    非严格意义上来讲，下图中红框框起来的部分便可以理解为一个滤波器，即带着一组固定权重的神经元。多个滤波器叠加便成了卷积层。

**多个滤波器叠加便成了卷积层**

### 学习sift,spin image,HoG,RIFT,GLOH,Textons

人工特征：**sift,spin image,HoG,RIFT,GLOH,Textons**

SIFT，即尺度不变特征变换（Scale-invariant feature transform，SIFT），是用于[图像处理](https://baike.baidu.com/item/%E5%9B%BE%E5%83%8F%E5%A4%84%E7%90%86" \t "https://baike.baidu.com/item/SIFT/_blank)领域的一种描述。这种描述具有尺度不变性，可在图像中检测出关键点，是一种局部特征描述子

第一阶段：SIFT特征的生成，即从多幅图像中提取对尺度缩放、旋转、亮度变化无关的特征向量。

第二阶段：SIFT特征向量的匹配

**Spin image**的思想是将一定区域的点云分布转换成二维的spin image​，然后对场景和模型的spin image​s进行相似性度量。原理图如下

深度学习：

**Hog：**方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在**[计算机视觉](http://lib.csdn.net/base/computervison" \t "http://blog.csdn.net/liulina603/article/details/_blank" \o "计算机视觉知识库)**和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。Hog特征结合SVM分类器已经被广泛应用于图像识别中，尤其在行人检测中获得了极大的成功。需要提醒的是，HOG+SVM进行行人检测的方法是法国研究人员Dalal在2005的CVPR上提出的，而如今虽然有很多行人检测**[算法](http://lib.csdn.net/base/datastructure" \t "http://blog.csdn.net/liulina603/article/details/_blank" \o "算法与数据结构知识库)**不断提出，但基本都是以HOG+SVM的思路为主

**LBP**是一种图像纹理特征（提取算法），它是一种局部特征，是照片分类和人脸检索研究中采用较多的特征提取算法之一。

特征提取算法是模式识别算法的重要组成部分之一，另外一个重要部分是分类算法。

模式识别领域，LBP是众多常用特征描述子之一，它的优势是定义简单、扩展方便、效率较高，在很多问题中的实验效果也不错。在图像物体识别领域，其它比较常用的特征描述子包括：Histogram of Oriented Gradient (HOG)，SIFT，SURF，Wavelet，Gabor，DCT等，具体选择哪种算子需要根据目标对象决定。

在人脸识别领域，像楼上所说，LBP和Gabor是公认效果较好的两组特征。当然，影响识别精度的因素很多，如：预处理、特征选择、分类器设计和后处理等。

# 729

## 1卷积神经网络的历史

1962年Hubel和Wiesel通过对猫视觉皮层细胞的研究，提出了**感受野**的概念，

1984年日本学者Fukushima基于感受野概念提出的**神经认知机(**)可以看作是卷积神经网络的第一个实现网络，也是感受野概念在人工神经网络领域的首次应用

神经认知机将**一个视觉模式分解成许多子模式**（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统**模型化，**使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

通常神经认知机包含两类神经元，即承担**特征抽取的S-元和抗变形的C-元**。S-元中涉及两个重要参数，即**感受野与阈值参数，前者确定输入连接的数目，后者则控制对特征子模式的反应程度**。

## 2）卷积神经网络的网络结构

卷积神经网络是一个多层的神经网络，每层由多个二维平面组成，而每个平面由多个独立神经元组成。

图：卷积神经网络的概念示范：输入图像通过和三个可训练的滤波器和可加偏置进行卷积，滤波过程如图一，卷积后在C1层产生三个特征映射图，然后特征映射图中每组的四个像素再进行求和，加权值，加偏置，通过一个Sigmoid函数得到三个S2层的特征映射图。这些映射图再进过滤波得到C3层。这个层级结构再和S2一样产生S4。最终，这些像素值被光栅化，并连接成一个向量输入到传统的神经网络，得到输出。

一般地，C层为特征提取层，每个神经元的输入与前一层的局部感受野相连，并提取该局部的特征，一旦该局部特征被提取后，它与其他特征间的位置关系也随之确定下来；S层是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射为一个平面，平面上所有神经元的权值相等。特征映射结构采用影响函数核小的sigmoid函数作为卷积网络的激活函数，使得特征映射具有位移不变性。

此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层（C-层）都紧跟着一个用来求局部平均与二次提取的计算层（S-层），这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。

### 滤波器

在CNN中,**滤波器filter(带着一组固定权重的神经元)**对局部输入数据进行卷积计算。每计算完一个数据窗口内的局部数据后,数据窗口不断平移滑动,直到计算完所有数据

## 3LeNet的模型建立及实现。

http://blog.csdn.net/niuwei22007/article/details/48950347

**输入层到s1是卷积运算**；输入层输入数据batch\_size个样本，每个样本是一个28∗28的图片，s1层滤波器大小为5∗5，滤波器为20个。   
　　经过卷积运算，s1的特征图数目为20个，每个特征图的大小为(28−5+1)∗(28−5+1)=24∗24。  
　　   
　　下面我们计算一下s1层需要训练的参数个数：s1生成有20个特征图，每个特征图的滤波器不同，同一个特征图的滤波器相同（这不废话吗）每个特征图含24∗24个元素，每个元素的生成是由本图的滤波器与原图像卷积计算得出。本图的滤波器含有5∗5个W元素+1个b元素(共5∗5+1=26个参数)。那么总的计算就是：共有 20∗(5∗5+1)=520个参数；20∗(24∗24)∗(5∗5+1)=299520个连接。

**s1层到c1层是下采样层。**计算过程就是将s1的输出数据作为c1的输入数据。因此，输入数据为20个24∗24的特征图。经过采样大小为2∗2感受野的计算后，结果仍然是20个特征图，每个特征图变为了12∗12。   
　　   
**滤波器的移动是有重叠的。而下采样是无重叠的**。最终原图是24∗24的，经过下采样之后，就变成12∗12了（就是行和列分别是原来的一半。因此整体就是原来的1/4）。总体就是在特征图数目不变的前提下，**将每个特征图进行了类似压缩的操作，这样元素个数大大降低，这就是所谓的下采样。**

同样我们计算一下c1层需要训练的参数个数：c1仍然生成20个特征图，每个特征图含12∗12个元素，每个元素的生成是由原图的2∗2区域元素累加，然后乘以1个可训练参数+1个偏置b参数。相当于本图的滤波器含有1∗1个W元素+1个b元素(共1∗1+1=2个参数)。那么总的计算就是：共有20∗(1∗1+1)=40个参数；20∗(12∗12)∗(2∗2+1)=14400个连接。。

　　c1到s2是卷积运算；输入是batch\_size个样本，每个样本是20个12∗12的特征图，s2层是50个滤波器，每个滤波器5∗5大小。然后经过卷积运算，生成50个8∗8的特征图。【c1与s2的全连接，但尚不清楚是如何计算的，因此未能算出其参数和连接。从其他资料经验看，应该有20∗50∗(5∗5+1)=26000个参数；20∗50∗(8∗8)∗(5∗5+1)=1664000个连接。】

　　s2层到c3层是下采样层，输入是50个8∗8特征图，采用2∗2感受野，结果是50个4∗4特征图。

**c3层到输出层经过一个MLP层。输入数据是将c3的4维输出转换为2维矩阵，即把后三维50∗4∗4转换为行数据，送入MLP的隐层，激活函数是tanh，输出是batch\_size；**然后再把结果送入MLP的LR层，经过softmax函数，将结果分为10类。

# 学习路径：

**matlab 的 Deep Learning 的工具箱 DeepLearnToolbox，这**里实现的 CNN 与其他最大的差别是采样层没 有权重和偏置，仅仅只对卷积层进行一个采样过程，这个工具箱的测试数据集是 MINIST，

每张图像是 28\*28 大小，它实现的是下面这样一个 CNN：

# 7 参考文献及推荐阅读

1. [人工神经网络wikipedia](http://www.wikiwand.com/zh-cn/%E4%BA%BA%E5%B7%A5%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
2. [斯坦福机器学习公开课](http://52opencourse.com/139/coursera%E5%85%AC%E5%BC%80%E8%AF%BE%E7%AC%94%E8%AE%B0-%E6%96%AF%E5%9D%A6%E7%A6%8F%E5%A4%A7%E5%AD%A6%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0%E7%AC%AC%E5%85%AB%E8%AF%BE-%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C%E7%9A%84%E8%A1%A8%E7%A4%BA-neural-networks-representation" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
3. [http://neuralnetworksanddeeplearning.com/](http://neuralnetworksanddeeplearning.com/" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
4. 雨石 卷积神经网络：[http://blog.csdn.net/stdcoutzyx/article/details/41596663](http://blog.csdn.net/stdcoutzyx/article/details/41596663" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
5. cs231n 神经网络结构与神经元激励函数：[http://cs231n.github.io/neural-networks-1/](http://cs231n.github.io/neural-networks-1/" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)，[中译版](https://zhuanlan.zhihu.com/p/21462488" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
6. cs231n 卷积神经网络：[http://cs231n.github.io/convolutional-networks/](http://cs231n.github.io/convolutional-networks/" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
7. 七月在线寒老师讲的**5月dl班第4次课CNN与常用框架视频**，已经剪切部分放在七月在线官网：[julyedu.com](https://www.julyedu.com/video/play/42/206" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
8. 七月在线5月深度学习班第5课CNN训练注意事项部分视频：[https://www.julyedu.com/video/play/42/207](https://www.julyedu.com/video/play/42/207" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
9. 七月在线5月深度学习班：[https://www.julyedu.com/course/getDetail/37](https://www.julyedu.com/course/getDetail/37" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
10. 七月在线5月深度学习班课程笔记——No.4《CNN与常用框架》：[http://blog.csdn.net/joycewyj/article/details/51792477](http://blog.csdn.net/joycewyj/article/details/51792477" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
11. 七月在线6月数据数据挖掘班第7课视频：数据分类与排序
12. 手把手入门神经网络系列(1)\_从初等数学的角度初探神经网络：[http://blog.csdn.net/han\_xiaoyang/article/details/50100367](http://blog.csdn.net/han_xiaoyang/article/details/50100367" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
13. 深度学习与计算机视觉系列(6)\_神经网络结构与神经元激励函数：[http://blog.csdn.net/han\_xiaoyang/article/details/50447834](http://blog.csdn.net/han_xiaoyang/article/details/50447834" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
14. 深度学习与计算机视觉系列(10)\_细说卷积神经网络：[http://blog.csdn.net/han\_xiaoyang/article/details/50542880](http://blog.csdn.net/han_xiaoyang/article/details/50542880" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
15. zxy 图像卷积与滤波的一些知识点：[http://blog.csdn.net/zouxy09/article/details/49080029](http://blog.csdn.net/zouxy09/article/details/49080029" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
16. zxy 深度学习CNN笔记：[http://blog.csdn.net/zouxy09/article/details/8781543/](http://blog.csdn.net/zouxy09/article/details/8781543/" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
17. [http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/](http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)，[中译版](http://www.csdn.net/article/2015-11-11/2826192" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
18. 《神经网络与深度学习》中文讲义：[http://vdisk.weibo.com/s/A\_pmE4iIPs9D](http://vdisk.weibo.com/s/A_pmE4iIPs9D" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
19. ReLU与sigmoid/tanh的区别：[https://www.zhihu.com/question/29021768](https://www.zhihu.com/question/29021768" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
20. CNN、RNN、DNN内部网络结构区别：[https://www.zhihu.com/question/34681168](https://www.zhihu.com/question/34681168" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
21. 理解卷积：[https://www.zhihu.com/question/22298352](https://www.zhihu.com/question/22298352" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
22. 神经网络与深度学习简史：[1 感知机和BP算法](http://mp.weixin.qq.com/s?__biz=MzA3MzI4MjgzMw==&mid=402032673&idx=1&sn=d7e636b6d033cbcf8a74dfaf710e9ccf&scene=21" \l "wechat_redirect" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)、[4 深度学习的伟大复兴](http://chuansong.me/n/2523251" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
23. 在线制作gif 动图：[http://www.tuyitu.com/photoshop/gif.htm](http://www.tuyitu.com/photoshop/gif.htm" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
24. [支持向量机通俗导论（理解SVM的三层境界）](http://blog.csdn.net/v_july_v/article/details/7624837" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank)
25. [CNN究竟是怎样一步一步工作的？](http://www.jianshu.com/p/fe428f0b32c1" \t "http://m.blog.csdn.net/v_JULY_v/article/details/_blank) 本博客把卷积操作具体怎么个计算过程写清楚了，但这篇把为何要卷积操作也写清楚了，而且配偶图非常形象，甚赞。

# 学习caffe

**脚本运行必须在caffe文件夹的根目录下运行**

1.2. Caffe代码层次。  
回答里面有人说熟悉Blob，Layer，Net，Solver这样的几大类，我比较赞同。我基本是从这个顺序开始学习的，这四个类复杂性从低到高，贯穿了整个Caffe。把他们分为三个层次介绍。

* **Blob：**是基础的数据结构，是用来保存学习到的参数以及网络传输过程中产生数据的类。
* **Layer：**是网络的基本单元，由此派生出了各种层类。修改这部分的人主要是研究特征表达方向的。
* **Net：**是网络的搭建，将Layer所派生出层类组合成网络。**Solver：**是Net的求解，修改这部分人主要会是研究DL求解方向的

你可以多跑跑几个例子，熟悉一下环境和接口

我觉得最好的方法是通过单步调试的方式跟着程序一步一步的在网络里前向传播，然后再被当成误差信息传回来

１．学习程序的第一步，先让程序跑起来，看看结果，这样就会有直观的感受。  
Caffe的官网上[Caffe | Deep Learning Framework](https://link.zhihu.com/?target=http://caffe.berkeleyvision.org/" \t "_blank) 提供了很多的examples,你可以很容易地开始训练一些已有的经典模型，如LeNet。我建议先从 [LeNet MNIST Tutorial](https://link.zhihu.com/?target=http://caffe.berkeleyvision.org/gathered/examples/mnist.html" \t "_blank)开始，因为数据集很小，网络也很小但很经典，用很少的时间就可以跑起来了。当你看到terminal刷拉拉的一行行输出，看到不断减少的loss和不断上升的accuracy，训练结束你得到了99+%的准确率，感觉好厉害的样子。你可以多跑跑几个例子，熟悉一下环境和接口。  
  
２．单步调试，跟着Caffe在网络里流动  
当玩了几天之后，你对Caffe的接口有点熟悉了，对已有的例子也玩腻了，你开始想看看具体是怎么实现的了。我觉得最好的方法是通过单步调试的方式跟着程序一步一步的在网络里前向传播，然后再被当成误差信息传回来。  
  
Caffe就像一个你平常编程中Project,你可以使用IDE或者GDB去调试它，这里我们不细说调试的过程。你可以先跟踪前向传播的过程，无非就是从高层次到低层次的调用Forward函数，　Solver->Net->Layer->Specific Layer (Convolution等...).后向传播也类似，但因为你对Caffe里面的各种变量运算不熟悉，当你跟踪完前向传播时可能已经头晕眼花了，还是休息一下，消化一下整个前向传播的流程。  
  
刚刚开始你没有必要对每个Layer的计算细节都那么较真，大概知道程序的运算流程就好，这样你才可以比较快的对Caffe有个大体的把握。  
  
３．个性化定制Caffe  
到这里，你已经可以说自己有用过Caffe了，但是还不能算入门，因为你还不知道怎么修改源码，满足自己特定的需求。我们很多时候都需要自己定义新的层来完成特定的运算，这时你需要在Caffe里添加新的层。  
  
你一开肯定无从下手，脑子一片空白。幸运的是Caffe github上的Wiki [Development · BVLC/caffe Wiki · GitHub](https://link.zhihu.com/?target=https://github.com/BVLC/caffe/wiki/Development" \t "_blank)已经有了教程了，而且这是最接近latest Caffe的源码结构的教程，你在网上搜到的Blog很多是有点过时的，因为Caffe最近又重构了代码。你可以跟着它的指导去添加自己的层。  
  
虽然你已经知道要在哪里添加自己的东西了，但你遇到最核心的问题是如何写下面这四个函数。

* forward\_cpu()
* forward\_gpu()
* backward\_cpu()
* backward\_gpu()

你可以先模仿已有的层去实现这四个函数，而且我相信forward函数很快就可以写出来了，但backward的还是一头雾水。**这时我们就要补补神经网络里最核心的内容了——Backpropagation.**  
  
４．理解并实现Backpropagation  
这个我觉得是与平台无关的，不管你是使用Caffe、Torch 7,还是Theano,你都需要深刻理解并掌握的。因为我比较笨，花了好长时间才能够适应推导中的各种符号。其实也不难，就是误差顺着Chain rule法则流回到前面的层。我不打算自己推导后向传播的过程，因为我知道我没有办法将它表达得很好，而且网上已经有很多非常好的教程了。下面是我觉得比较好的学习步骤吧。

* 从浅层的神经网络（所谓的全连接层）的后向传播开始，因为这个比较简单，而且现在我们常说的CNN和LSTM的梯度计算也最终会回归到这里。
  + 第一个必看的是Ng深入浅出的Ufldl教程[UFLDL Tutorial](https://link.zhihu.com/?target=http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial" \t "_blank)，还有中文版的，这对不喜欢看英语的同学是个好消息。当然你看一遍不理解，再看一遍，忘了，再看，读个几遍你才会对推导过程和数学符号熟悉。我头脑不大行，来来回回看了好多次。
  + 当然，Ufldl的教程有点短，我还发现了一个讲得更细腻清晰的教程, [Michael Nielsen](https://link.zhihu.com/?target=http://michaelnielsen.org" \t "_blank)写的[Neural networks and deep learning](https://link.zhihu.com/?target=http://neuralnetworksanddeeplearning.com/chap2.html" \t "_blank)。它讲得实在太好了，以至于把我的任督二脉打通了。在Ufldl的基础上读这个，你应该可以很快掌握全连接层的反向传播。
  + 最后在拿出standford大牛karpathy的一篇博客[Hacker's guide to Neural Networks](https://link.zhihu.com/?target=http://karpathy.github.io/neuralnets/" \t "_blank)，这里用了具体的编程例子手把手教你算梯度，并不是推导后向传播公式的，是关于通用梯度计算的。用心去体会一下。
* 这时你跃跃欲试，回去查看Caffe源码里Convolution层的实现，但发现自己好像没看懂。虽说卷积层和全连接层的推导大同小异，但思维上还是有个gap的。我建议你先去看看Caffe如何实现卷积的，Caffe作者贾扬清大牛在知乎上的回答[在 Caffe 中如何计算卷积？](https://www.zhihu.com/question/28385679)让我茅塞顿开。重点理解im2col和col2im.
* 这时你知道了Convolution的前向传播，还差一点就可以弄明白后向传播怎么实现了。我建议你死磕Caffe中Convolution层的计算过程，把每一步都搞清楚，经过痛苦的过程之后你会对反向传播有了新的体会的。在这之后，你应该有能力添加自己的层了。再补充一个完整的添加新的层的教程[Making a Caffe Layer • Computer Vision Enthusiast](https://link.zhihu.com/?target=http://chrischoy.github.io/blog/research/making-caffe-layer/" \t "_blank)。这篇教程从头开始实现了一个Angle To Sine Cosine Layer，包含了梯度推导，前向与后向传播的CPU和GPU函数，非常棒的一个教程。
* 最后，建议学习一下基本的GPU Cuda编程，虽然Caffe中已经把Cuda函数封装起来了，用起来很方便，但有时还是需要使用kernel函数等Cuda接口的函数。这里有一个入门的视频教程，讲得挺不错的[NVIDIA CUDA初级教程视频](https://link.zhihu.com/?target=http://www.iqiyi.com/a_19rrhbvoe9.html" \t "_blank)

真的需要时间的，，而且怎么办。。

要了解函数的具体使用吗

有点难入门，在window下面的教程不多，，比较难，机器学习？？每天投入这件事情，，效率。。

# 8.4

## 步幅和填充

https://www.zhihu.com/question/52668301

## ReLU（修正线性单元）层

在每个卷积层之后，通常会立即应用一个非线性层（或激活层）。其目的是给一个在卷积层中刚经过线性计算操作（只是数组元素依次（element wise）相乘与求和）的系统**引入非线性特征**。过去，人们用的是像双曲正切和 S 型函数这样的非线性方程，但研究者发现 ReLU 层效果好得多，

**如果没有relu**

在这种情况下你每一层输出都是上层输入的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机（Perceptron）了。线性变换模型capacity是很有限的，连最简单的异或问题都没法解决。没有激活函数的每层都相当于矩阵相乘。就算你叠加了若干层之后，无非还是个矩阵相乘罢了

## 池化层作用

池化层往往跟在卷积层后面。通过平均池化或者最大池化的方法将之前卷基层得到的特征图做一个聚合统计。假设L层的卷积层得到的某一特征图有100\*100这么大的尺寸。选一个2\*2的区域做不重叠的最大池化，池化层会输出50\*50那么大的图，达到降低数据量的目的

## 网络过程

## 数据转换

### 图像数据转换成db（leveldb/lmdb)文件

<http://www.cnblogs.com/denny402/p/5082341.html>

在caffe中，作者为我们提供了这样一个文件：convert\_imageset.cpp，存放在根目录下的tools文件夹下。编译之后，生成对应的可执行文件放在 buile/tools/ 下面，这个文件的作用就是用于将图片文件转换成caffe框架中能直接使用的db文件。

该文件的使用格式：

convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME

需要带四个参数：

FLAGS: 图片参数组，后面详细介绍

ROOTFOLDER/: 图片存放的绝对路径，从linux系统根目录开始

LISTFILE: 图片文件列表清单，一般为一个txt文件，一行一张图片

DB\_NAME: 最终生成的db文件存放目录

如果图片已经下载到本地电脑上了，那么我们首先需要创建一个图片列表清单，保存为txt

本文以caffe程序中自带的图片为例，进行讲解，图片目录是  example/images/, 两张图片，一张为cat.jpg, 另一张为fish\_bike.jpg，表示两个类别。

# 8.5

<http://blog.csdn.net/u010540396/article/details/52895074>

简单起见，我们的代码选用一层卷积层

1池化（pooling）采用平均2\*2 %

2网络结点数说明： %

输入层：28\*28 %

第一层：24\*24（卷积）\*20 % tanh %

第二层：12\*12（pooling）\*20 %

第三层：100(全连接) %

第四层：10(softmax) %

3、网络训练部分采用800个样本，检验部分采用100个样本

### 1卷积

%实现卷积层操作,实现点乘

function [state]=convolution(data,kernel)

### 2网络初始化

layer\_c1\_num=20;

layer\_s1\_num=20;

layer\_f1\_num=100;

layer\_output\_num=10;

%权值调整步进

yita=0.01;

%bias初始化？？？

bias\_c1=(2\*rand(1,20)-ones(1,20))/sqrt(20);

bias\_f1=(2\*rand(1,100)-ones(1,100))/sqrt(20);

%卷积核初始化？？？每个维数都要初始化

[kernel\_c1,kernel\_f1]=init\_kernel(layer\_c1\_num,layer\_f1\_num);

%pooling核初始化

pooling\_a=ones(2,2)/4;

%全连接层的权值

weight\_f1=(2\*rand(20,100)-ones(20,100))/sqrt(20);

weight\_output=(2\*rand(100,10)-ones(100,10))/sqrt(100);

disp('网络初始化完成......');

## Softmax

<https://www.zhihu.com/question/23765351>

## 2神经网络和深度学习

### 1感知机

感知机如何**赋予不同evidence权重**来达到做出决策的目的。一个由感知机构成的复杂网络能够做出更加精细的决策

第一列感知机——通常称为第一层感知机——通过赋予输入的evidence权重，做出三个非常简单的决策。第二层感知机呢？每一个第二层感知机通过赋予权重给来自第一层感知机的决策结果，来做出决策。通过这种方式，第二层感知机可以比第一层感知机做出更加复杂以及更高层次抽象的决策。第三层感知机能够做出更加复杂的决策。通过这种方式，**一个多层网络感知机可以做出更加精细的决策。**

### 2sigmoid神经元

防止权重和偏置改变后轻易改变结果

与感知机有些相似，但做了一些修改使得我们在轻微改变其权值和偏置时只会**引起小幅度的输出变化**。这是使由sigmoid神经元构成的网络能够学习的关键因素。

### 3递归神经网络

到目前为止，我们讨论的都是前馈神经网络（feedforward neural networks），即把上一层的输出作为下层输入的神经网络。这种网络是不存在环的——信息总是向前传播，从不反向回馈。如果我们要制造一个环，那么我们将会得到一个使函数输入依赖于其输出的网络。这很难去理解，所以我们并不允许存在这样的环路。

但是，我们也有一些存在回馈环路可能性的人工神经网络模型。这种模型被称为[递归神经网络（recurrent neural networks）](http://en.wikipedia.org/wiki/Recurrent_neural_network" \t "https://hit-scir.gitbooks.io/neural-networks-and-deep-learning-zh_cn/content/chap1/_blank)。该模型的关键在于，神经元在变为非激活态之前会在一段有限时间内均保持激活状态。这种激活状态可以激励其他的神经元，被激励的神经元在随后一段有限时间内也会保持激活状态。如此就会导致更多的神经元被激活，一段时间后我们将得到一**个级联的神经元激活系统。**在这个模型中环路并不会带来问题，因为神经元的输出只会在一段之间之后才影响到它的输入，它并非实时的。

### 4偏置

bias和weight是不同的加权形式，组合起来就是linearity。你可以看作是weight2。有些paper为了描述简便，会省略掉。去掉bias，拟合效果会变差。

bias之后的linearity接non-linearity，怎么会分歧。bias初始值为0就行了，alexnet里似乎发现1有助于加速训练，我实际测的时候没有这效果。

把阀值转化为一个权值，这样阀值都是0，也好调整和实现。

### 5遗传算法

**遗传算法**（英语：genetic algorithm (GA) ）是[计算数学](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%95%B0%E5%AD%A6" \o "计算数学)中用于解决[最佳化](https://zh.wikipedia.org/wiki/%E6%9C%80%E4%BD%B3%E5%8C%96" \o "最佳化)的搜索[算法](https://zh.wikipedia.org/wiki/%E7%AE%97%E6%B3%95" \o "算法)，是[进化算法](https://zh.wikipedia.org/wiki/%E8%BF%9B%E5%8C%96%E7%AE%97%E6%B3%95" \o "进化算法)的一种。进化算法最初是借鉴了[进化生物学](https://zh.wikipedia.org/wiki/%E8%BF%9B%E5%8C%96%E7%94%9F%E7%89%A9%E5%AD%A6" \o "进化生物学)中的一些现象而发展起来的，这些现象包括[遗传](https://zh.wikipedia.org/wiki/%E9%81%97%E4%BC%A0" \o "遗传)、[突变](https://zh.wikipedia.org/wiki/%E7%AA%81%E5%8F%98" \o "突变)、[自然选择](https://zh.wikipedia.org/wiki/%E8%87%AA%E7%84%B6%E9%80%89%E6%8B%A9" \o "自然选择)以及[杂交](https://zh.wikipedia.org/wiki/%E6%9D%82%E4%BA%A4" \o "杂交)等。

遗传算法通常实现方式为一种[计算机模拟](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%A8%A1%E6%8B%9F" \o "计算机模拟)。对于一个最优化问题，一定数量的[候选解](https://zh.wikipedia.org/w/index.php?title=%E5%80%99%E9%80%89%E8%A7%A3&action=edit&redlink=1" \o "候选解（页面不存在）)（称为个体）可抽象表示为[染色体](https://zh.wikipedia.org/wiki/%E6%9F%93%E8%89%B2%E9%AB%94_(%E9%81%BA%E5%82%B3%E6%BC%94%E7%AE%97%E6%B3%95)" \o "染色体 (遗传算法))，使[种群](https://zh.wikipedia.org/wiki/%E7%A7%8D%E7%BE%A4" \o "种群)向更好的解进化。传统上，解用[二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E8%BF%9B%E5%88%B6" \o "二进制)表示（即0和1的串），但也可以用其他表示方法。进化从完全[随机](https://zh.wikipedia.org/wiki/%E9%9A%8F%E6%9C%BA" \o "随机)个体的种群开始，之后一代一代发生。在每一代中评价整个种群的[适应度](https://zh.wikipedia.org/wiki/%E9%80%82%E5%BA%94%E5%BA%A6" \o "适应度)，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群

### 6随机梯度下降（代码？

梯度下降法（gradient descent）能够帮助我们去**学习权重和偏置**

有种叫做随机梯度下降（SGD）的算法能够用来加速学习过程。想法就是通过随机选取小量输入样本来计算，进而可以计算。采取少量样本的平均值可以快速地得到梯度**，这会加速梯度下降过程，进而加速学习过程。**

更准确地说，SGD是**随机地选取小量的个训练数据**。我们将选取的这些训练数据标号，并把它们称为一个mini-batch。我们选取的要足够大才能保证的平均值才能接近所有样本的平均值，也就是说， 其中，第二个求和是针对整个训练数据集的。交换两边我们可以得到 证实了我们可以通过计算随机选取的mini-batch的梯度来估计整体的梯度。

为了将这种想法更明确地联系到神经网络的学习中来，假设用和分别代表权重和偏置。SGD就是随机地选取大小为一个mini-batch的训练数据，然后去训练这些数据，

### 7反向传播算法

我们没有讨论如何计算代价函数的梯度。这真是一个巨大的跳跃！在本章中我会介绍一个快速计算梯度的算法，就是广为人知的反向传播算法

反向传播算法的核心是一个偏微分表达式，**表示代价函数对网络中的权重（或者偏置）求偏导。**这个式子告诉我们，当我**们改变权重和偏置的时候，代价函数的值变化地有多快。**尽管这个式子有点复杂，这个式子也是很漂亮的，它的每一个部分都有自然的，直觉上的解释。因此，反向传播不仅仅是一种快速的学习算法，它能够让我们详细深入地了解改变权重和偏置的值是如何改变整个网络的行为的。这是非常值得深入学习的

反向传播算法的目标是计算代价函数对神经网络中出现的所有权重和偏置的偏导数和

### 8交叉熵

交叉熵衡量的是我们在知道yy的真实值时的平均**「出乎意料」程度**。当输出是我们期望的值，我们的「出乎意料」程度比较低；当输出不是我们期望的，我们的「出乎意料」程度就比较高。当然，我没有准确说明「出乎意料」是什么意思，所以这个措辞听起来很空洞。但事实上是有一种精确的信息理论方法来阐述「出乎意料」所表达的意思的。

在神经网络的计算当中，我们经常需要计算按照神经网络的正向传播计算的分数S1，和按照正确标注计算的分数S2，之间的差距，计算Loss，才能应用反向传播。**Loss定义为交叉熵**

我们能够从图像看出当**神经元输出接近时，曲线变得非常平坦，**因此就会变得非常小。等式（55）和等式（56）能告诉我们和会变得很小。这就是**学习速度变慢的根源。**我们如何来避免这种减速呢？事实证明我们可以用不同的代价函数比如交叉熵（cross-entropy）代价函数来替代平方代价函数。

# 8.10

## Max Pooling 和mean pooling

对于mean pooling，真的是好简单：假设pooling的窗大小是2x2, 在forward的时候啊，就是在前面卷积完的输出上依次不重合的取2x2的窗平均，得到一个值就是当前mean pooling之后的值。backward的时候，把一个值分成四等分放到前面2x2的格子里面就好了。如下

forward: [1 3; 2 2] -> [2]

backward: [2] -> [0.5 0.5; 0.5 0.5]

max pooling就稍微复杂一点，forward的时候你只需要把2x2窗子里面那个最大的拿走就好了，backward的时候你要把当前的值放到之前那个最大的位置，其他的三个位置都弄成0。如下

forward: [1 3; 2 2] -> 3

backward: [3] -> [0 3; 0 0]

## 2toolbox的cnn

在Toolbox中，CNN的实现过程主要包括下面几个文件

cnntrain.m //用于训练CNN的m文件

cnntest.m //用于预测的m文件

cnnsetup.m //设置训练参数的m文件

cnnnumgradcheck.m //用于梯度检查的m文件

cnnff.m //用于计算前向过程的m文件

cnnbp.m //用于计算误差方向传递的m文件

cnnapplygrads.m //用于使用梯度下降算法的m文件

在test\_example\_CNN.m中，Toolbox的作者使用了MNIST手写字体数据库对CNN部分进行了测试。我将从test\_example\_CNN.m文件开始，按照使用CNN的过程对CNN的每一部分代码进行学习。

t**est\_example\_CNN.m**

将读取训练样本和测试样本；

将样本恢复为图像，并归一化为0~1；

设置CNN网络的层数，每一层的结构，学习速率，batch的大小，迭代的次数；

调用cnnsetup函数，初始化CNN的结构和参数；

调用cnntrain函数，训练CNN；

调用cnntest函数，对测试样本进行预测； 输出错误率。

**cnnsetup.m**

从cnnsetup.m文件可以看出，CNN结构中每一层的特征图的输入特征图的个数，以及每一层的可训练参数都在cnnsetup函数中进行设置。这里和经典的LeNet-5中使用的CNN结构不同之处在于，对于下采样层，并没有设置可训练的系数参数。

**cnntrain.m**

在cnntrain.m文件中，主要是调用了cnnff进行前向计算，然后调用cnnbp函数进行误差计算，最后调用cnnappplygrads函数更新cnn的参数。

Cnff.m

在cnnff.m文件中，我们可以看出，对于下采样层，这个实现并没有使用到在cnnsetup中初始化的偏置b，同时，下采样层也没有经过sigmoid函数进行激活。这个与经典的LeNet-5的实现之间有较大区别，同时，在这个实现中，对于卷积层的每一个特征图，都是将前一层中所有的特征图分别进行卷积后全部相加，然后加上一个偏置后，经过Sigmoid函数激活后得到的，并没有经过选择前一层的某些特征图进行卷积的过程。这一点和LeNet-5的实现也有所不同

**cnnbp.m**

在cnnbp.m文件中，主要计算了各层之间的delta，以及卷积层的参数的梯度，由于在cnnff函数中，没有用到下采样层的偏置，所以，在cnnbp函数中，并没有计算该参数的梯度的过程。

**Cnnapplaygrads.m**

在cnnapplaygrads函数中，主要是使用梯度下降算法更新卷积层的参数和输出层的参数，由于在cnnff中并没有使用到下采样层的偏置这个参数，所以在cnnapplaygrads函数中，也没有更新这个参数的过程。

**Cnntest.m**

cnntest函数主要用来对测试样本进行预测，并计算错误率，其实，对于CNN来说，对样本进行预测就是简单地使用cnnff函数计算样本的前向过程，然后找出输出层中值最大的那个神经元的类别，就是样本对应的类别，所以，在cnntest函数中，并没有什么重要的内容

**在看完CNN的代码之后，就可以开始动手修改这些代码了。Enjoy it~**

### 小tip

深度学习Matlab DeepLearningToolBox 工具包最常见错误解决办法：

 错误：assert(~isOctave() || compare\_versions(OCTAVE\_VERSION, ‘3.8.0‘, ‘>=‘), [‘Octave 3.8.0 or greater is  required for CNNs as there is a bug in convolution in previous......

isOctave这个语句是为了抛出一个程序在Octave平台上运行时的一个BUG，在matlab平台上可以直接注释掉

# 811

## 1python用caffe，还是有挺多不懂，很好

<http://blog.csdn.net/Jesse_Mx/article/details/58605385>

net = caffe.Net(model\_def, *#* **定义模型结构deploy.prototxt**

model\_weights, *#* **包含了模型的训练权值bvlc\_reference\_caffenet.caffemodel**caffe.TEST) *#* **使用测试模式(不执行dropout)？？**

**设置输入图像大小？？**

*# 加载ImageNet图像均值 (随着Caffe一起发布的)？？？？*mu = np.load(caffe\_root + **'python/caffe/imagenet/ilsvrc\_2012\_mean.npy'**)  
mu = mu.mean(1).mean(1) *# 对所有像素值取平均以此获取BGR的均值像素值***print 'mean-subtracted values:'**, zip(**'BGR'**, mu)  
  
 *# 对输入数据进行变换 ？？？？caffe.io.*Transformertransformer = caffe.io.Transformer({**'data'**: net.blobs[**'data'**].data.shape})  
  
transformer.set\_transpose(**'data'**, (2, 0, 1)) *# 将图像的通道数设置为outermost的维数*transformer.set\_mean(**'data'**, mu) *# 对于每个通道，都减去BGR的均值像素值*transformer.set\_raw\_scale(**'data'**, 255) *# 将像素值从[0,255]变换到[0,1]之间*transformer.set\_channel\_swap(**'data'**, (2, 1, 0)) *# 交换通道，从RGB变换到BGR  
  
# 设置输入图像大小*net.blobs[**'data'**].reshape(50, *# batch 大小 ？？？？*net.blobs3, *# 3-channel (BGR) images* 227, 227) *# 图像大小为:227x227  
# 3.2 加载图像(caffe自带的)并进行预处理。caffe.io.load\_image*image = caffe.io.load\_image(caffe\_root + **'examples/images/cat.jpg'**)  
transformed\_image = transformer.preprocess(**'data'**, image)  
plt.imshow(image)  
plt.show()  
  
*# 将图像数据拷贝到为net分配的内存中*net.blobs[**'data'**].data[...] = transformed\_image  
  
*### 执行分类*output = net.forward()  
output\_prob = output[**'prob'**][0] *# batch中第一张图像的概率值***print 'predicted class is:'**, output\_prob.argmax()

**2**

*# 加载ImageNet标签，确定类别*labels\_file = caffe\_root + **'data/ilsvrc12/synset\_words.txt'** *# if not os.path.exists(labels\_file): !../data/ilsvrc12/get\_ilsvrc\_aux.sh*labels = np.loadtxt(labels\_file, str, delimiter=**'\t'**) *#分割***print 'output label:'**, labels[output\_prob.argmax()]

# 812-818

## 生成txt和lmdb，训练

<http://blog.csdn.net/u010417185/article/details/52119863>

Python格式均值计算

<http://www.cnblogs.com/denny402/p/5102328.html>

训练过程

<http://blog.csdn.net/deeplearninglc007/article/details/40086503>

测试：

$ ./build/tools/caffe.bin test -model=examples/mnist/lenet\_train\_test.prototxt -weights=examples/mnist/lenet\_iter\_10000.caffemodel -gpu=0

## 1测试cifar图像：

*811okcaffe+python.py*

<http://www.mamicode.com/info-detail-1319332.html>

## 0转换成lmdb

### 1二进制到lmdb

测试图片和标签转换成二进制：

D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\data\mnist\812testmnist的readjpg.m

二进制转换成lmdb

create\_mnist.bat

..\..\Build\x64\Release\convert\_mnist\_data.exe train-images.idx3-ubyte train-labels.idx1-ubyte..\..\examples\mnist\mnist\_train\_lmdb

### 2图像到lmdb，用python？？/convert\_imageset

利用convert\_imageset

**1创建图片清单**

create\_filelist.sh

# /usr/bin/env sh

DATA=examples/images

echo "Create train.txt..."

rm -rf $DATA/train.txt

find $DATA -name \*cat.jpg | cut -d '/' -f3 | sed "s/$/ 1/">>$DATA/train.txt

find $DATA -name \*bike.jpg | cut -d '/' -f3 | sed "s/$/ 2/">>$DATA/tmp.txt

cat $DATA/tmp.txt>>$DATA/train.txt

rm -rf $DATA/tmp.txt

echo "Done.."

create\_lmdb.sh

#!/usr/bin/en sh

DATA=examples/images

rm -rf $DATA/img\_train\_lmdb

build/tools/convert\_imageset --shuffle \--resize\_height=256 --resize\_width=256 \/home/xxx/caffe/examples/images/ $DATA/train.txt $DATA/img\_train\_lmdb

得到db文件

## 1

3、create\_mnist812.bat，mnist二进制生成lmdb

..\..\Build\x64\Release\convert\_mnist\_data.exe test\_img\_ubyte test\_lable\_ubyte ..\..\examples\mnist\mnist\_test\_lmdb

1. mnist\_test.bat

## 2利用classification.exe对自己的手写数字进行分类

### 1、准备若干张手写数字，二值化

Classification：proto、model、mean、words.txt、jpg

在D:\Study\Caffe\caffe-windows\examples\mnist目录下，同时将需要测试的图片也放到这里。

### 新建****test\_personwrite.bat****

D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\Build\x64\Release\classification.exe D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\examples\mnist\lenet.prototxt D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\examples\mnist\lenet\_iter\_10000.caffemodel D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\examples\mnist\mean.binaryproto D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\examples\mnist\lable.txt D:\AnZhuangbao\caffe\719\caffe-master716\caffe-master\examples\mnist\39.jpg

pause

## 4均值文件生成与使用，

利用caffe的compute\_image\_mean.exe

利用mnist\_mean.bat

Build\x64\Release\compute\_image\_mean.exe examples\mnist\mnist\_train\_lmdb mean.binaryproto --backend=lmdb

Pause

# 815

## [SVM和SoftMax的原理区别对比](http://blog.csdn.net/u011534057/article/details/51451958) cs231

多类别支持向量机损失

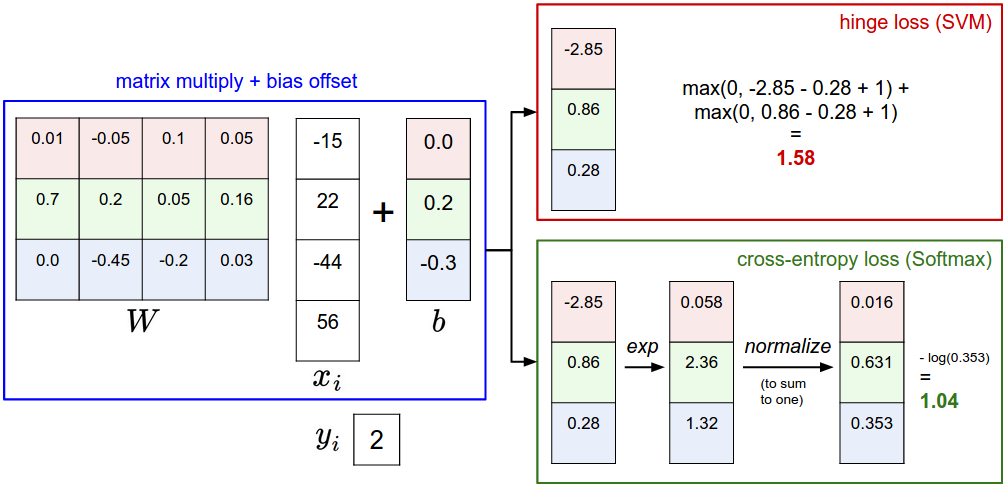
如果要用一句精简的话来描述它，就是它(SVM)希望正确的类别结果获得的得分比不正确的类别，至少要高上一个固定的大小Δ。

最常见的**正则化项**是L2范数，它会对幅度很大的特征权重给很高的惩罚：

w1=[1,0,0,0]，w2=[0.25,0.25,0.25,0.25]。平均的平方更小，所以w2好

准确地说，SVM分类器使用hinge loss(有时候也叫max-margin loss)。而Softmax分类器使用互熵损失/cross-entropy loss。Softmax分类器从softmax函数(恩，其实做的事情就是把一列原始的类别得分归一化到一列和为1的正数表示概率)得到，softmax函数使得互熵损失可以用起来。

### 1比较损失函数



我们先用一张图来表示从输入端到分类结果，SVM和Softmax都做了啥：

区别就是拿到原始像素数据映射得到的得分之后的处理，而正因为处理方式不同，我们定义不同的损失函数，有不同的优化方法

### 2.6.1 另外的差别

* SVM下，我们能完成类别的判定，但是实际上我们得到的类别得分，大小顺序表示着所属类别的排序，但是得分的绝对值大小并没有特别明显的物理含义。
* Softmax分类器中，结果的绝对值大小表征属于该类别的概率。

举个例子说，SVM可能拿到对应 猫/狗/船 的得分[12.5, 0.6, -23.0]，同一个问题，Softmax分类器拿到[0.9, 0.09, 0.01]。这样在SVM结果下我们只知道『猫』是正确答案，而在Softmax分类器的结果中，我们可以知道属于每个类别的概率。

但是，Softmax中拿到的概率，其实和正则化参数λ有很大的关系，会考虑每个值

**因此svm具有稳定的附加性**

## 2libsvm安装

**<http://blog.csdn.net/abcjennifer/article/details/7370177>**

**>> [predict\_label,accuracy,decision\_values]=svmpredict(heart\_scale\_label,heart\_scale\_inst,model)**

**Accuracy = 86.6667% (234/270) (classification)**

>> [predict\_label] = svmpredict(heart\_scale\_label,heart\_scale\_inst,model)

## 3libsvm使用

<http://blog.sina.com.cn/s/blog_5dd2e9270100wrfa.html>

## 4Matcaffe使用

classfy\_demo的 matlab运行不了：要把release路径加入matlab的路径

[scores, maxlabel] = classification\_demo(im, 0); %0是cpu，1是gpu，1会崩溃

先运行classfy——demo没有错误然后用

[scores, maxlabel] = classification\_demo(im, 0); 试一下

<http://blog.csdn.net/kkk584520/article/details/49475633>

<http://blog.csdn.net/sinat_31821401/article/details/69945442?locationNum=2&fps=1>

Minust调试caffe

<http://blog.csdn.net/zb1165048017/article/details/52447109#reply>

### 2训练500张图片预测

<http://blog.csdn.net/baidu_30028771/article/details/60882620>

### 1生成txt，matlab或者脚本

<http://www.cnblogs.com/denny402/p/5083300.html>

### 3[solver.prototxt参数说明](http://blog.csdn.net/u010417185/article/details/52182833)

olver算是caffe的核心的核心，它协调着整个模型的运作。caffe程序运行必带的一个参数就是solver配置文件。**运行代码**一般为

# caffe train --solver=\*\_slover.prototxt

在Deep Learning中，往往loss function是非凸的，没有解析解，我们需要通过优化方法来求解。solver的主要作用就是交替调用前向（forward)**[算法](http://lib.csdn.net/base/datastructure" \t "http://blog.csdn.net/u010417185/article/details/_blank" \o "算法与数据结构知识库)**和后向（backward)算法来更新参数，从而最小化loss，实际上就是一种迭代的优化算法。

到目前的版本，caffe提供了六种优化算法来求解最优参数，在solver配置文件中，通过设置type类型来选择，接下来，我们先来看一个实例：

net: "examples/mnist/lenet\_train\_test.prototxt" #设置深度网络模型。每一个模型就是一个net

test\_iter: 100

test\_interval: 500

base\_lr: 0.01

momentum: 0.9

type: SGD

weight\_decay: 0.0005

lr\_policy: "inv"

gamma: 0.0001

power: 0.75

display: 100

max\_iter: 20000

snapshot: 5000

snapshot\_prefix: "examples/mnist/lenet"

solver\_mode: CPU

## 特征融合

[特征提取-特征后期融合-多核学习方法（MKL）](http://blog.csdn.net/wuhang19911024/article/details/46287639)

<http://blog.csdn.net/wuhang19911024/article/details/46287639>

**SVM：支持向量机，**目前在分类中得到广泛的应用

特征融合：主要用来描述各种不同的特征融合方式，常见的方式有前期融合，就是前面所描述的将各个特征拼接在一起，后期融合本文后面会提到

核函数：SVM遇到线性不可分问题时，可以通过核函数将向量映射到高维空间，在高维空间线性可分

多核学习：在利用SVM进行训练时，会涉及核函数的选择问题，譬如线性核，rbf核等等，多核即为融合几种不同的核来训练

图像的分类问题跟一般的分类问题方法本质上没太大差异，主要差异体现在特征的抽取上以及特征的计算时间上

图像特征的抽取分为两部分，一部分是针对**通用图像的特征**，还有一部分则是针对**特定类别抽取的特征**。这些特征与普通的文本特征不一致的地方在于，一个图像特征由于存在分块、采样、小波变换等**，可能维度就已经很高**。。在分类过程中，如果将这些特征拼接在一起直接就可能过千维，但是实际在标注样本时，人工标注的正负样本也才几千张，所以在选择分类器时，**挑选svm，该**分类器由于可以在众多分类面中选择出最优分界面，以及在小样本的学习中加入惩罚因子产生一定软边界，可以有效规避overfitting。

在特征的计算时间上，由于图像处理涉及的矩阵计算过多，一个特征的计算时间慢的可以达到0.3秒，所以如何挑选出既有效又快速的特征也非常重要。

**交叉验证**是用来干嘛的？验证调优参数呗，全部特征融合在一起，再来调，尽管验证时间长，不要紧，反正模型是离线训练的，多调会也没关系

Svm的分类函数形似上是类似于一个神经网络，输出由中间若干节点的线性组合构成，而多核学习的分类函数则类似于一个比svm更高一级的神经网络，其输出即为中间一层核函数的输出的线性组合。

# 817

## 提取特征：caffe的工具

**$TOOL/extract\_features.bin $MODEL $PROTOTXT $LAYER $LEVELDB $BATCHSIZE**

build/tools/extract\_features.bin examples/imagenet/caffe\_reference\_imagenet\_model

examples/\_temp/imagenet\_val.prototxt fc7 examples/\_temp/features 10

其中 fc7 是最高层的特征，我们也可以使用其他层提取，像 conv5 或 pool3

**备注：用 caffe 的 python 接口提取和保存特征到text文本下**

features = net.blobs['conv5'].data # 提取卷积层 5 的特征

np.savetxt('conv5\_feature.txt', features) # 将特征存储到本文文件中

**现在Caffe的Matlab接口 (matcaffe3) 和python接口都非常强大, 可以直接提取任意层的feature map以及parameters, 所以本文仅仅作为参考, 更多最新的信息请参考:**

<http://blog.csdn.net/jiandanjinxin/article/details/50410290>

**例子**

#!/usr/bin/env sh

# args for EXTRACT\_FEATURE

TOOL=../../build/tools

MODEL=../../examples/imagenet/caffe\_reference\_imagenet\_model #下载得到的caffe model

PROTOTXT=../../examples/\_temp/imagenet\_val.prototxt # 网络定义

LAYER=conv1 # 提取层的名字，如提取fc7等

LEVELDB=../../examples/\_temp/features\_conv1 # 保存的leveldb路径

BATCHSIZE=10

# args for LEVELDB to MAT

DIM=290400 # 需要手工计算feature长度

OUT=../../examples/\_temp/features.mat #.mat文件保存路径

BATCHNUM=1 # 有多少哥batch， 本例只有两张图， 所以只有一个batch

$TOOL/extract\_features.bin $MODEL $PROTOTXT $LAYER $LEVELDB $BATCHSIZE

python **leveldb2mat.py** $LEVELDB $BATCHNUM $BATCHSIZE $DIM $OUT

# [Caffe提取任意层特征并进行可视化](http://www.cnblogs.com/platero/p/3967208.html)

# 818训练

## 1. 总结lenet\_solver.prototxt中的各参数含义

1、一次迭代是指取出64副图片进行训练，所以10000次迭代就需要640000副图片，但是MNIST训练集中不是就60000副图片吗？（图片就是指28x28的字符）

一次迭代是64副图,10000次迭代(64万次)超过60000副图片, 会重复从这60000副图片取

caffe可以设置每个epoch 重新shuffle一遍训练数据,这样子效果应该会更好一些,可以参考[关于caffe里面image\_data\_layer的使用 - 博客频道 - CSDN.NET](https://link.zhihu.com/?target=http://blog.csdn.net/kuaitoukid/article/details/51323940" \t "_blank)

2,一个epoch是把60000副图片都训练一遍, batch\_size: 64 的话,迭代60000/64次就一个epoch,一般要几十个epoch就差不多收敛了吧

iteration： 数据进行一次前向-后向的训练  
batchsize: 每次迭代训练图片的数量  
epoch: 1个epoch就是将所有的训练图像全部通过网络训练一次  
  
例如：例子中1280000张图片，batchsize=256,则1个epoch需要1280000/256=5000次iteration它的max-iteration=450000,则共有450000/5000=90个epoch,而lr什么时候衰减与stepsize有关，减少多少与gamma有关，即:若stepsize=500, base-lr=0.01, gamma=0.1,则当迭代到第一个500次时，lr第一次衰减，衰减lr=lr\*gamma=0.01\*0.1=0.001,以后重复该过程，所以stepsize是lr的衰减步长，gamma是lr的衰减系数。在训练过程中，每到一定的迭代次数都会test net,  
这里的迭代次数是由test-interval决定的，如test=1000,则每迭代1000次测试一遍网络，而test-size, test-iter, 和test图片的数量决定了怎样test, test-size决定了test时每次迭代输入图片的数量，test-iter就是test所有的图片的迭代次数，如：500张test图片，test-iter=100则test-size=5, 而solver文档里只需要根据test图片总数量来设置test-iter，以及根据需要设置test-interval即可。

## [caffe中mnist中 lenet\_train\_test.prototxt和lenet.prototxt（deploy文件）区别](http://blog.csdn.net/redfivehit/article/details/72902418)

两个文件都在examples/mnist 中, lenet\_train\_test.prototxt 文件是设置train、test的网络，deploy文件是分类**[测试](http://lib.csdn.net/base/softwaretest" \t "http://blog.csdn.net/redfivehit/article/details/_blank" \o "软件测试知识库)**加载的文件。

大体区别思路是网络配置文件会规定的详细一些，deploy文件只是告诉分类程序（mnist\_classification.bin）网络结构是怎么样的，不需要反向计算，不需要计算误差

学习教程

<http://www.linuxidc.com/Linux/2016-11/136774p3.htm>

定制solve参数

## [实用：使用caffe训练模型时solver.prototxt中的参数设置解析](http://blog.csdn.net/jiongnima/article/details/68929303)

## [train loss与test loss结果分析](http://blog.csdn.net/smf0504/article/details/71698354)

test\_iter：batch\_size \*test\_iter=测试集总数

（1）batchsize：批大小。在深度学习中，一般采用SGD训练，即每次训练在训练集中取batchsize个样本训练；

（2）iteration：1个iteration等于使用batchsize个样本训练一次；

（3）epoch：1个epoch等于使用训练集中的全部样本训练一次；

## 819查看训练日志

<http://blog.csdn.net/zb1165048017/article/details/52653501>

%tmp%查找temp文件夹看以caffe.exe开头的文件

权值微调

<http://blog.csdn.net/qiuqinjun/article/details/52608381>

## softmax vs. softmax-loss

用户可能最终目的就是得到各个类别的概率似然值，这个时候就只需要一个 Softmax层，而不一定要进行softmax-Loss 操作；或者是用户有通过其他什么方式已经得到了某种概率似然值，然后要做最大似然估计，此时则只需要后面的 softmax-Loss 而不需要前面的 Softmax 操作。因此提供两个不同的 Layer 结构比只提供一个合在一起的 Softmax-Loss Layer 要灵活许多。

### 5Dropout

Dropout是hintion最近2年提出的；为了防止模型过拟合，Dropout可以作为一种trikc供选择。在hinton的论文摘要中指出，在每个训练批次中**，通过忽略一半的特征检测器（让一半的隐层节点值为0），可以明显地减少过拟合现象。**这种方式可以减少特征检测器间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。

1.Dropout是在标准的bp网络的的结构上，使bp网的隐层激活值，以一定的比例v变为0，即按照一定比例v，随机地让一部分隐层节点失效；在后面benchmark实验测试时，部分实验让隐层节点失效的基础上，使输入数据也以一定比例（试验用20%）是部分输入数据失效（这个有点像denoising autoencoder），这样得到了更好的结果。

2.去掉权值惩罚项，取而代之的事，限制权值的范围，给每个权值设置一个上限范围；如果在训练跟新的过程中，权值超过了这个上限，则把权值设置为这个上限的值（这个上限值得设定作者并没有说设置多少最好，后面的试验中作者说这个上限设置为15时，最好；为啥？估计是交叉验证得出的实验结论）。

这样处理，不论权值更新量有多大，权值都不会过大。此外，还可以使算法使用一个比较大的学习率，来加快学习速度，从而使算法在一个更广阔的权值空间中搜索更好的权值，而不用担心权值过大

## 训练过程

**1matlab修改路径生成mat，生产图片（改地址，改load）**

**2sh生产lmdb（修改count和目录）**

**3用bat，修改prototxt训练模型**

**改了一下stride，test**

# 821

## -Visualizing and Understanding Convolutional Networks论文

### 1用反卷积网络可视化整个卷积网络，并进行分析和调优。

<http://www.gageet.com/2014/10235.php>

在每个卷积层都加上了一个反卷积层。在卷积、ReLU、Max-pooling之后，不仅输出给下一层用作输入，也为反卷积提供输入。而反卷积层**依次进行unpooling、ReLU和反卷积**

Unpooling：在max-pooling中利用**switches**表格记录每个最大值的位置，然后在unpooling中奖该位置填回最大数值，其余位置填0。

ReLU：直接利用ReLU函数，仍然确保输出为非负数。

反卷积：利用相同卷积核的转置作为核，与输入做卷积运算

## 论文-用cnn做回归

肿瘤网络-肺结核特征使用多视角卷积神经网络结合高斯过程

end-to-end trainable multi-view deep Convolutional Neural Network (CNN) for nodule characterization

增加样本数量-缩放，旋转，噪声

首先，我们使用中值强度投影来获得对应于每个维度的2D补丁。 然后将三个图像级联以形成张量，其中图像用作输入图像的不同通道。

经过训练的网络用于从**输入图像中提取特征**，然后进行**高斯过程（GP）回归以获得恶性分数**。

我们还经验性地确定了**不同高度结节属性（如钙化，球形度等）对恶性肿瘤测定的意义。**这些属性被发现是与深层多视图CNN特征的互补，并且获得了比其他方法的显着改进。

2发现鉴别成像特征

以前手工特征：histograms , (SIFT) (LBP) and (HOG) 然后用svm分类器，随机森林RF

Cnn:we perform an end-to-end training of CNN for nodule characterization while combining multi-view features to obtain improved characterization performance.

与这些方法形成鲜明对比的是，我们对组合多视点特征进行结节表征的CNN的端对端训练，以获得改进的表征性能。

•我们从零开始对CNN进行端到端的培训，以实现神经网络的全部潜力，即学习辨别特征。

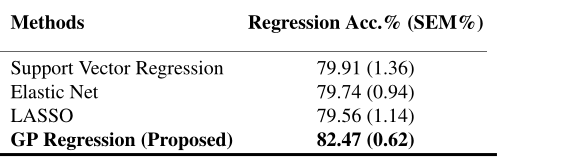
•对包含超过1000个低剂量CT扫描的肺结节的数据集进行广泛的实验评估。

•我们经验性地确定了高水平结节属性（如钙化，分叶，球形度等）与深层CNN特征的补充意义，以改善恶性肿瘤的判定。

增加数据：用高斯，噪声，旋转等，，，

1提取第一个全连接层的特征，然后用GP回归，+-1范围内被认为正确

都用cnn特征来做回归



2结合高水平特征+cnn，噶水平特征容易被检测

calcification, spiculation, lobulation, margin, sphericity, and texture.

不同类别的噪音

就是回归，通过不同的噪音获取许多数据，然后进行回归

### 高斯回归

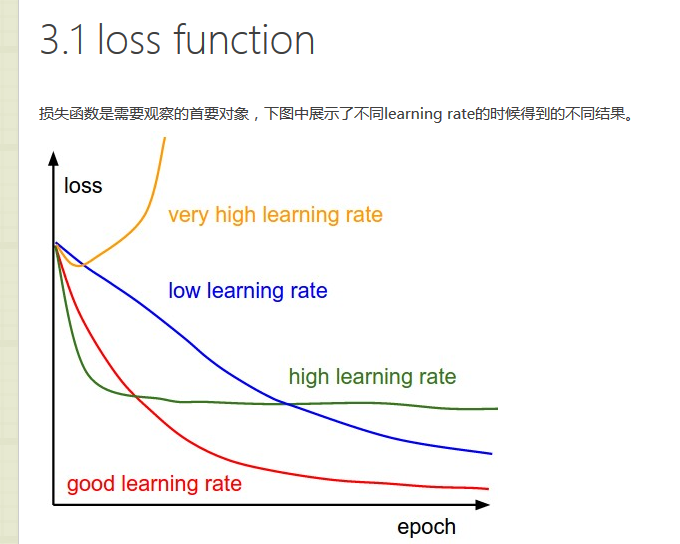
斯过程回归(GPR)和贝叶斯线性回归类似，区别在于高斯过程回归中用核函数代替了贝叶斯线性回归中的基函数（其实也是核函数，线性核

<https://wenku.baidu.com/view/fb36bd03d4d8d15abe234eb2.html>

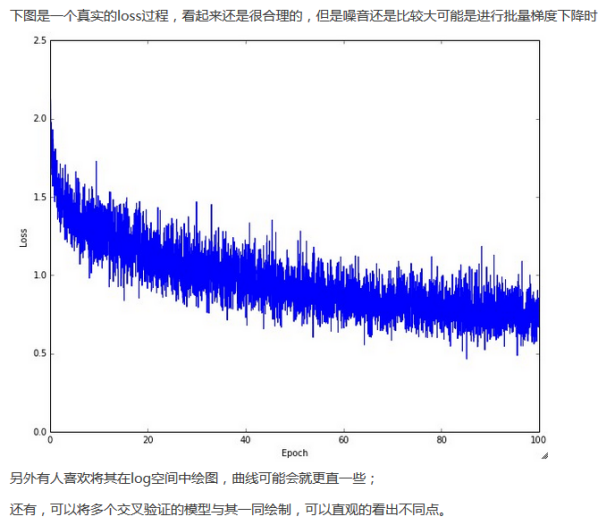
## 论文2：成像生物标志物发现肺癌生存预测

# 调整参数9.2

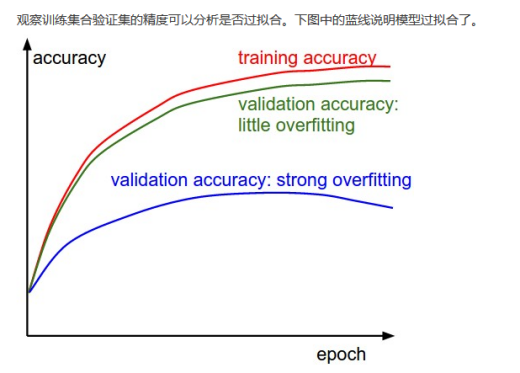
## 学习率调整

****

## Batch调整

****

## 3过拟合

****

# 9.3从神经网络说起：深度学习初学者不可不知的25个术语和概念

原文链接：<http://www.afenxi.com/post/44887>

**神经网络基础概念：**

## 神经元—

正如神经元是大脑的基本单位一样，在神经网络结构中，神经元也是一个小单位。大家不妨想象一下当我们接触到新的信息时，大脑是如何运作的。

首先，我们会在脑中处理这个信息，然后产生输出信息。在神经网络中也是如此，神经元接收到一个输入信息，然后对它进行加工处理，然后产生输出信息，传输到其他神经元中进行进一步信息处理。

## 权重—

—当输入信息到达神经元时，它就会乘上一个权重。举例来说，如果一个神经元包含两个输入信息，那么每个输入信息都被赋予它的关联权重。我们随机初始化权重，并在模型训练过程中更新这些权重。

接受训练后的神经网络会赋予它认为重要的输入信息更高的权重值，而那些不重要的输入信息权重值则会相对较小。权重值为零就意味着这个特征是无关紧要的。

我们不妨假设输入信息为a，其关联权重为IMG_257,通过节点后，输入信息变为

IMG_258：

## 偏置 ——

除了权重之外，输入还有另一个线性分量，被称为偏置。输入信息乘上权重后再加上偏置，用来改变权重乘输入的范围。加上偏置之后，结果就变为

IMG_260，这就是输入信息变换的最终线性分量。

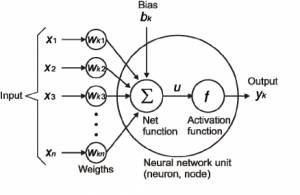
## ****激活函数****—

线性分量应用可以到输入信息，非线性函数也可以应用到输入信息。这种输入信息过程是通过激活函数来实现的。

激活函数将输入信号翻译成输出信号。激活函数产生的输出信息为IMG_261，其中f(x)就是激活函数。

在下面的图表中，我们可以看到，输入信息数量为n，表示为IMG_262到IMG_263，相应的权重为IMG_264到IMG_265。将偏置设为IMG_266。权重乘以输入信息，再加偏置，我们设所得的值为u：u=∑w\*x+b

将u带入激活函数中，最后我们就可以得到从神经元输出的IMG_267

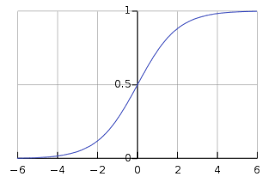


**常用激活函数**

最常用的激活函数有Sigmoid、ReLU 和softmax。

* Sigmoid——Sigmoid是最常用的激活函数之一。 它的定义为：

IMG_269

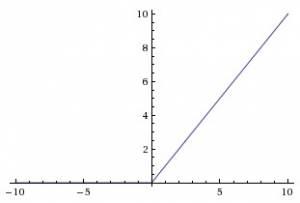


Sigmoid函数会生成0到1之间的更平滑的取值范围。我们可能需要观察输出值的变化，同时输入值也会略有变化。而平滑的曲线更方便我们观察，因此它优于阶梯函数（step functions）。

* ReLU（线性修正单位）——不同于sigmoid函数，现在的网络更倾向于使用隐层ReLu激活函数。该函数的定义是：

IMG_271

当X> 0时，函数的输出为X，X <= 0时为0。该函数如下所示：



使用ReLU的好处主要是它对于大于0的所有输入值都有对应的不变导数值。而常数导数值可以加快对网络的训练。

* Softmax——Softmax激活函数常用于输出层的分类问题。 它与sigmoid函数类似，唯一的区别是在Softmax激活函数中，输出被归一化，总和变为1。

如果我们遇到的是二进制输出问题，就可以使用Sigmoid函数，而如果我们遇到的是多类型分类问题，使用softmax函数可以轻松地为每个类型分配值，并且可以很容易地将这个值转化为概率。

## ****MLP (多层神经网络)**** –—

—MLP(多层神经网络) – 单一神经元无法执行高度复杂的任务。因此，需要大量神经元聚集在一起才能生成我们所需要的输出信号。

最简单的网络由一个输入层、一个输出层、一个隐含层组成，每一层上都有多个神经元，并且每一层上的神经元都和下一层上的神经元连接在了一起，这样的网络也被称为全互连网络(fully connected networks)。

## ****正向传播(Forward Propagation)**** –

——正向传播指的是输入信号通过隐藏层传递到输出层的传递过程。

在正向传播中，信号仅沿单一方向向前正向传播，输入层将输入信号提供给隐藏层，隐藏层生成输出信号，这一过程中没有任何反向移动。

## （9）成本函数(Cost Function) –—

—当我们建立一个网络后，网络将尽可能地使输出值无限接近于实际值。我们用成本函数（或损失函数）来衡量该网络完成这一过程的准确性。成本函数（或损失函数）将在该网络出错时，予以警告。

运行网络时，我们的目标是：尽可能地提高我们的预测精度、减少误差，由此最小化成本函数。最优化的输出即当成本函数（或损失函数）为最小值时的输出。

若将成本函数定义为均方误差，则可写成：

IMG_276

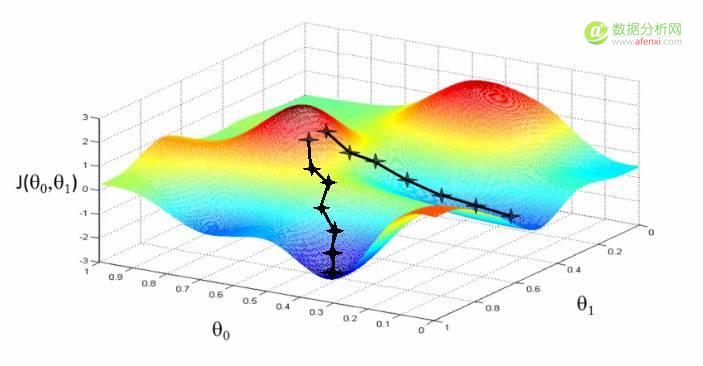
m在这里是训练输入值(training inputs)，a 是预计值，y是特定事例中的实际值。

学习过程围绕着如何最小化成本。

## ****梯度下降(Gradient Descent)**** –——

梯度下降是一种优化算法，以最小化成本。想象一下，当你下山时，你必须一小步一小步往下走，而不是纵身一跃跳到山脚。

因此，我们要做的是:比如，我们从X点开始下降，我们下降一点点，下降ΔH，到现在的位置，也就是X-ΔH，重复这一过程，直到我们到达“山脚”。“山脚”就是最低成本点。



从数学的角度来说，要找到函数的局部极小值，须采取与函数梯度负相关的“步子”，即：梯度下降法是用负梯度方向为搜索方向的，梯度下降法越接近目标值，步长越小，前进越慢。

## ****学习速率 (Learning Rate)**** –——

学习率指每次迭代中对成本函数的最小化次数。简单来说，我们把下降到成本函数最小值的速率称为学习率。选择学习率时，我们必须非常小心，学习率既不应过大——会错过最优解，也不应过小——使网络收敛将需要很多很多步、甚至永不可能。

## ****反向传播(Back propagation)**** –—

—在定义一个神经网络的过程中， 每个节点会被随机地分配权重和偏置。

一次迭代后，我们可以根据产生的结果计算出整个网络的偏差，然后用偏差结合成本函数的梯度，对权重因子进行相应的调整，使得下次迭代的过程中偏差变小。

这样一个结合成本函数的梯度来调整权重因子的过程就叫做反向传播。在反向传播中，信号的传递方向是朝后的，误差连同成本函数的梯度从输出层沿着隐藏层传播，同时伴随着对权重因子的调整。

15)**Dropout方法** —— Dropout是一个可以阻止网络过拟合的规则化方法。就像它的名字那样，在训练过程中隐藏的某些特定神经元会被忽略掉(drop)。

这意味着网络的训练是在几个不同的结构上完成的。这种dropout的方式就像是一场合奏，多个不同结构网络的输出组合产生最终的输出结果。

16) **分批标准化 (Batch Normalization)** –——分批标准化就像是人们在河流中用以监测水位的监察站一样。

这是为了保证下一层网络得到的数据拥有合适的分布。在训练神经网络的过程中，每一次梯度下降后权重因子都会得到改变，从而会改变相应的数据结构。

但是下一层网络希望能够得到与之前分布相似的数据，因此在每一次数据传递前都需要对数据进行一次正则化处理。

**卷积神经网络**

17) **过滤器/滤波器 (Filters)** ——CNN中的滤波器，具体是指将一个权重矩阵乘以输入图像的一个部分，产生相应的卷积输出。

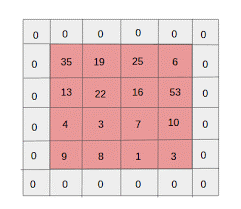
比方说，对于一个28×28的图片而言，将一个3×3的滤波器与图片中3×3的矩阵依次相乘，从而得到相应的卷积输出。

滤波器的尺寸通常比原始图片要小，与权重相似，在最小化成本的反向传播中，滤波器也会被更新。就像下面这张图片一样，通过一个过滤器，依次乘以图片中每个3×3的分块，从而产生卷积的结果。

## ****卷积神经网络CNN (Convolutional neural network)****—

—卷积神经网络通常用来

20) **补白 (Padding)**补白(Padding)通常是指给图像的边缘增加额外的空白，从而使得卷积后输出的图像跟输入图像在尺寸上一致，这也被称作相同补白(Same Padding)。



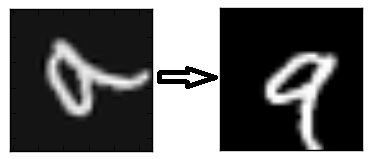
如应用过滤器，在相同补白的情况下，卷积后的图像大小等于实际图像的大小。

有效补白(Valid Padding)指的是保持图片上每个真实的像素点，不增加空白，因此在经历卷积后数据的尺寸会不断变小。

21) **数据增强 (Data Augmentation)** –——数据增强指的是从已有数据中创造出新的数据，通过增加训练量以期望能够提高预测的准确率。

比如，在数字识别中，我们遇到的数字9可能是倾斜或旋转的，因此如果将训练的图片进行适度的旋转，增大训练量，那么模型的准确性就可能会得到提高。

通过“旋转”“照亮”的操作，训练数据的品质得到了提升，这种过程被称作数据增强。



**递归神经网络**

22) **递归神经元 (Recurrent NeuralNetwork)** —— 对于递归神经元来说，经由它自己处理过的数据会变成自身下一次的输入，这个过程总共会进行t次。

如下图所示，将递归神经元展开就相当于t个不同的神经元串联起来，这种神经元的长处是能够产生一个更全面的输出结果。

## 23) ****递归神经网络(RNN-Recurrent NeuralNetwork)****  ——

递归神经网络通常被用于处理序列化的数据，即前一项的输出是用来预测下一项的输入。

递归神经网络通常被用于处理序列化的数据，即前一项的输出是用来预测下一项的输入。递归神经网络中存在环的结构，这些神经元上的环状结构使得它们能够存储之前的数据一段时间，从而使得能够预测输出。

与递归神经元相似，在RNN中隐含层的输出会作为下一次的输入，如此往复经历t次，再将输出的结果传递到下一层网络中。这样，最终输出的结果会更全面，而且之前训练的信息被保持的时间会更久。

隐藏层将反向传递错误以更新权重。这被称为backpropagation through time (BPTT).

## ****梯度消失问题**** –

——当激活函数的梯度非常小时，会出现梯度消失问题。在反向传播过程中，权重因子会被多次乘以这些小的梯度。

因此会越变越小，随着递归的深入趋于“消失”， 使得神经网络失去了长程可靠性。这在递归神经网络中是一个较普遍的问题，对于递归神经网络而言，长程可靠性尤为重要。

这一问题可通过采用ReLu等没有小梯度的激活函数来有效避免。

## ****梯度爆炸问题**** –—

—梯度爆炸问题与梯度消失问题正好相反，梯度爆炸问题中，激活函数的梯度过大。

在反向传播过程中，部分节点的大梯度使得他们的权重变得非常大，从而削弱了其他节点对于结果的影响，这个问题可以通过截断(即设置一个梯度允许的最大值)的方式来有效避免