CS231n学习笔记

Assignment Git:

<https://github.com/CS231n-zju/CS231n>

视频地址：

<https://www.youtube.com/watch?v=vT1JzLTH4G4&index=2&list=PLe7764SJVnV10-Nr7e0sBlC9J0LRf4sQo>

课程作业：

<http://cs231n.github.io/>

Syllabus:

<http://cs231n.stanford.edu/syllabus.html>

Note翻译:

<http://www.52ml.net/17723.html>

**Week4**

[Convolutional Neural Networks: Architectures, Convolution / Pooling Layers 2](#_Toc1685790942)

[1.结构 2](#_Toc1419758538)

[2.layers 2](#_Toc167779281)

[3.Conv layer 2](#_Toc1654328369)

[3.1感受野(receptive field) 2](#_Toc1862667601)

[3.2四个超参数 2](#_Toc659769637)

[3.3权值共享的概念 3](#_Toc384282915)

[3.4计算实现举例 3](#_Toc1670445682)

[3.5 BP过程 3](#_Toc1555209335)

[3.6 1\*1的卷积 3](#_Toc121050832)

[3.7 扩张卷积 4](#_Toc242309773)

[4. 池化层 4](#_Toc1796633649)

[4.1计算举例 4](#_Toc488894352)

[4.2 BP 4](#_Toc1096265687)

[4.3 抛弃pooling层的趋势 4](#_Toc1885819421)

[5.归一化层和全连接层 4](#_Toc1228558788)

[6.卷积神经网络的结构 4](#_Toc772095927)

[6.1常见的模式 4](#_Toc1791323981)

[6.2 结构选择 5](#_Toc2042918956)

[6.3层的尺寸超参数调参 5](#_Toc174656566)

[7.知名的CNN结构 5](#_Toc234589344)

[8.计算内存占用时的trick 6](#_Toc149433465)

[Unknown word 6](#_Toc1755923635)

[Assignment2\_Q1: Fully-connected Neural Network 7](#_Toc992241500)

[1.目标 7](#_Toc1805706361)

[2.注意layer之间的forward和backward是怎么运行的 8](#_Toc2125418541)

[3.batch\_size/epoc/iteration的区别 8](#_Toc1387621504)

[4. Optimazation 8](#_Toc1163851423)

[Unknown word 9](#_Toc962713126)

[Assignment2\_Q2: Batch Normalization 9](#_Toc526916064)

[1.论文阅读Batch Normalization 9](#_Toc241625024)

[2.实现中的问题记录 10](#_Toc501020421)

[2.1Batch Normalization: alternative backward 10](#_Toc1946674602)

[2.2把batch normalization加入FCN 10](#_Toc409404306)

[3.baseline,benchmark,sota的区别 11](#_Toc7865142)

[Unknown word 11](#_Toc1661858555)

[Video5.Lecture 5 | Convolutional Neural Networks 11](#_Toc1069173943)

[Unknown word 11](#_Toc392148057)

# Convolutional Neural Networks: Architectures, Convolution / Pooling Layers

CNN的两个advantages:

1. Make the forward function more efficient to implement.
2. Reduce the amount of parameters in the network.

## 1.结构

A ConvNet is made up of Layers. Every Layer has a simple API:

It transforms an input 3D volume to an output 3D volume.

## 2.layers

包括三种:Convolutional Layer, Pooling Layer, and Fully-Connected Layer

最简单的双层CNN(一个隐含层): [INPUT - CONV - RELU - POOL - FC]

总的特点:

每一层都是从3D volume到3D volume的转换.

CONV/FC有参数学习(w,b), RELU/POOL没有.

CONV/FC/POOL 有超参数学习(learning rate等等), RELU没有.

维度trans过程,以32\*32\*3的raw pixel输入为例,选择12个filters,10个类别:

INPUT: [32x32x3]

CONV layer : [32x32x12]

RELU layer: [32x32x12]

POOL layer: [16x16x12](举例,反正width和height会下降)

FC (i.e. fully-connected) layer :[1x1x10]

## 3.Conv layer

Intuitively, the network will learn filters that activate when they see some type of visual feature. We will stack these activation maps(12 filters) along the depth dimension and produce the output volume.

CNN训练得到的是滤波器，本质上是对于某种特定的模式有响应，反之无响应.

### **3.1感受野**(receptive field)

一个神经元只和图像的一个区域连接,这个连接的大小叫做感受野.例如[32\*32\*3]的图,假设感受野是5\*5,那么每个神经元只看图像中5\*5的区域,每个神经元的参数是:5\*5\*3+1(偏置)

### 3.2四个超参数

深度,步长(stride),padding决定输出的尺寸.

* 感受野的大小
* 深度就是filters的个数.
* 步长控制感受窗口的速度,影响输出的长宽.
* Padding的目的主要是为了让输入输的长宽一致,不使用padding还可能使得图像的边缘信息迅速损失掉.

**公式:**

输入图像尺寸:W1\*H1\*D1

Filters的个数:K

感受野:F

Stride:S

Padding:P

输出的长/宽为:W2=(W1-F+2\*P)/S+1 , H2=(H1-F+2\*P)/S+1 , D2=K , b2=K(k个偏置)

ps:这里注意步长的设定要能够被(W-F+2\*P)整除!

### 3.3权值共享的概念

一个神经元只看输入图中的一个感受野区域!但是他们都用5\*5\*3的权值.

权值共享指的是一层filter上的神经元,共享5\*5\*3的这个感受野的权值!

12个filters,就有12个[5\*5]的感受野,12组权值.也就是(5\*5\*3+1)\*12=76\*12=912

BP的时候,把所有神经元的梯度累加,来更新这一个共享的filter的权重.

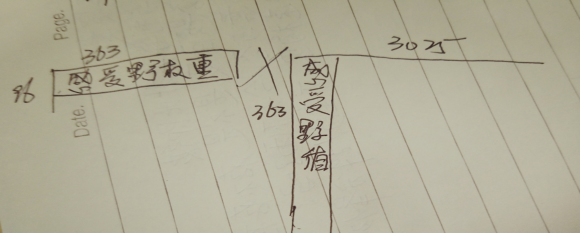
可以这么做的缘故是:图像具有结构上的平移不变性.但是在人脸中不太一样,有时候我们需要做局部连接.

### 3.4计算实现举例

W\_row = [96 x 363]

X\_col = [363 x 3025]

计算整个卷积层的输出(一共96层filters) : np.dot(W\_row, X\_col)



最后把结果reshape成[96\*55\*55]

### 3.5 BP过程

也是一个卷积!

### 3.6 1\*1的卷积

作用主要有两个,这里文中讲的不是很清楚,参考知乎:

https://www.zhihu.com/question/56024942/answer/154291405

1.降维（ dimension reductionality ）。比如，一张500 \* 500且厚度depth为100(一般是3 channels)的图片在20个filter上做1\*1的卷积，那么结果的大小为500\*500\*20。

2.加入非线性。卷积层之后经过激励层，1\*1的卷积在前一层的学习表示上添加了非线性激励（ non-linear activation ），提升网络的表达能力；

### 3.7 扩张卷积

可以增大感受野!

## 池化层

作用:降维,减少参数量,降低过拟合.

### 4.1计算举例

Accepts a volume of size W1×H1×D1W1×H1×D1

Requires two hyperparameters:

* their spatial extent F,
* the stride S,

Produces a volume of size W2×H2×D2 where:

* W2=(W1−F)/S+1
* H2=(H1−F)/S+1
* D2=D1

注意:

* Deep不变
* 不需要学习参数
* 一般不进行padding
* F值最好调小一些,常用F=2.
* Max polling的效果比average好

### 4.2 BP

注意BP过程其实是对max函数做BP,所以需要记录当时更大的那一个数,毕竟只有它贡献了梯度.

### 4.3 抛弃pooling层的趋势

GAN等不太适于用pooling,可以用增大步长的方法降低数据量,未来可能会逐步discard pooling层.

## 5.归一化层和全连接层

归一化层: 模仿生物神经网络中的抑制作用,in practice,贡献很小.

全连接层: 和前一层全连接,乘积加偏置.

论文:Fully Convolutional Networks for Semantic Segmentation

链接:<https://arxiv.org/abs/1411.4038>

FC->CNN:用一个和输入同等大小的卷积核即可实现.

优点:可以让卷积网络在一张更大的输入图片上滑动，得到每个区域的输出（这样就突破了输入尺寸的限制）.

## 6.卷积神经网络的结构

包括:卷积层/池化层/全连接层

### 6.1常见的模式

INPUT -> [[CONV -> RELU]\*N -> POOL?]\*M -> [FC -> RELU]\*K -> FC

PS:

* \* indicates repetition
* POOL? indicates an optional pooling layer
* N >= 0 (and usually N <= 3)
* M >= 0
* K >= 0 (and usually K < 3)

### 6.2 结构选择

选择几个小感受野的卷积层垛堞而不是一个大感受野的卷积层.

**举例**:

三个[3\*3]感受野的卷积层放在一起,最后第三层卷积层对输入图像的感受野是[7\*7]

(PS:假设所求感受野=F,则有[ (F-3)+1 ]-3+1=3 , 故而F=7)

相比直接用一个[7\*7]的感受野,将会有一些优点:

1. 多个卷积层用非线性函数提取了更多图像的特征
2. 减少了参数量

缺点是:BP的时候内存占用量高一些.

**In practice:**

直接下一个最好的网络,然后fine tune它!

### 6.3层的尺寸超参数调参

**卷积层:**

输入数据的尺度最好可以被2除很多次.

F小一些,3\*3或者5\*5常见,如果一定要用7\*7这么大的,一般只用在第一层卷积层上.

S=1.

P=(F−1)/2,这样可以使得输入输出数据尺度不变.

**Pooling层:**

F=2,一般不超过3

S=2

用max pooling

**一些参数选择问题:**

为什么用s=1?为了让卷积层只负责降低数据深度,而pooling层只负责降低数据长宽.

不使用padding可能使得图像的边缘信息迅速损失掉.

出于对GPU性能的考虑,应当做出一定的妥协,一般发生在第一层.

## 7.知名的CNN结构

LeNet:最早对CNN进行应用(1990),主要用来识别邮政编码

AlexNet:2012年ImageNet ILSVRC比赛冠军,更深,更大,第一次用多层卷积层来提取特征(以前都是Conv后面立刻跟一个pooling)

ZF Net:2013年ILSVRC冠军,就是对AlexNet调参,调高中间卷积层的面积,第一层的stride和filter的size调低.

GoogLeNet:2014年ILSVRC冠军.使用inception结构极大的降低了参数量.使用了平均pooling.

VGGNet:2014年ILSVRC亚军.网络的深度体现算法的优度.从头到尾全部使用的是3x3的卷积和2x2的pooling!缺点在于参数太多了.绝大多数都来源于第一层的全连接,后来发现把全连接去掉不会影响性能,这样参数就大大降低了.

ResNet:2015年ILSVRC冠军.It features special skip connections and a heavy use of batch normalization,没有全连接层.

av16180503评论ResNet:普通深度神经网络（plain network）如果深度太深，根据梯度下降算法，离输出层越远的隐含层的乘积项会越多（通常会越乘越小），导致这些层的神经元训练会越缓慢（如果计算机精度不够的话，可能还会变0），导致难以或无法训练 残差网络，则是在普通神经网络基础上，把两层或多层的神经元组成一个residual block，将输入block和从最后一层的输出相加再通过输出函数送进下一个block。这样的好处是，在梯度下降的计算上，偏导数变成了1＋原来的乘积项，就不会因为越乘越小，而导致远离最终输出的那层隐含层调参缓慢或甚至无法调参.

Vgg中,几乎全部的计算和内存消耗都发生在前面的卷积层,但是参数量最大的是后面的全连接层,这很常见.

## 8.计算内存占用时的trick

1.测试时抛弃前面层的激活值.(训练的时候不可以,要用来求梯度)

2.存储参数的内存往往要在总的参数量上\*3(参数值,参数的梯度,有一些bp的过程比如动量法还要多存一个cache)

3.在2的基础上还有一些额外的空间

最后的计算结果应该以GB为单位.把所有的参数量\*4(如果是double还要\*8),除以1024几次去得到MB,GB级的结果.如果发现内存不够,可以用减少batch\_size的方法,因为大多数内存消耗是在卷积层的激活值那里的.

## Unknown word

Entry 条目

Blotch 斑点

Honeycomb 蜂窝

Asymmetry 不对称

Neatly 整齐地

Mutual 相互的

Alleviate 减轻

Dilate 扩张

Preserved 保守的

Paradigm 范式

Intricate 错综复杂的

Fatigue 疲劳的

Aforementioned 上述的

Compromising 妥协

Bottlenecked 瓶颈

runner-up 亚军

Tweaking 调整

Miscellaneous 杂项

Heuristic 启发式

# Assignment2\_Q1: Fully-connected Neural Network

## 1.目标

之前实现的loss和gradient并不适用于大模型,接下来要做的是全连接网络的实现.每层各有一个forward和backward函数.

The forward function will receive inputs, weights, and other parameters and will return both an output and a cache object storing data needed for the backward pass, like this:

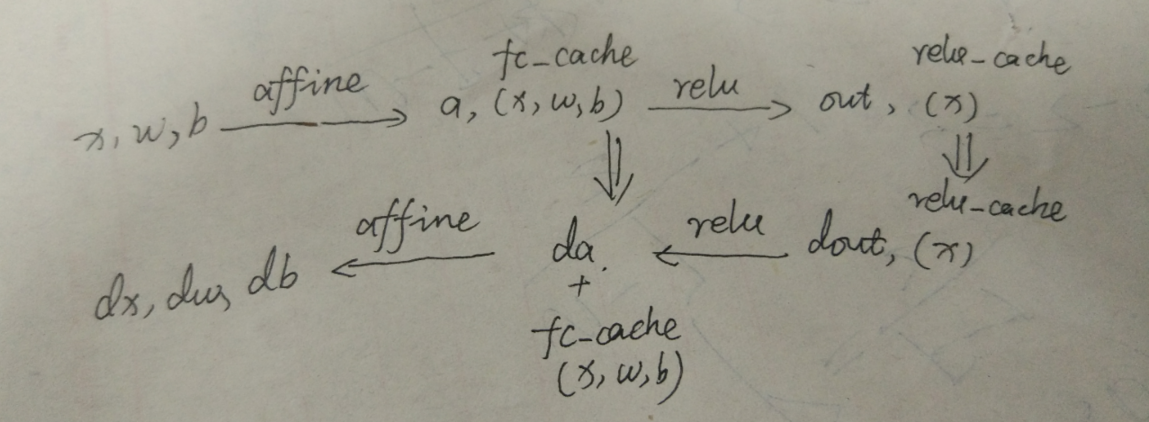
|  |
| --- |
| **def** layer\_forward(x, w):  """ Receive inputs x and weights w """  *# Do some computations ...*  z = *# ... some intermediate value*  *# Do some more computations ...*  out = *# the output*  cache = (x, w, z, out) *# Values we need to compute gradients*  **return** out, cache |

The backward pass will receive upstream derivatives and the cache object, and will return gradients with respect to the inputs and weights, like this:

|  |
| --- |
| **def** layer\_backward(dout, cache):  """ Receive derivative of loss with respect to outputs and cache, and compute derivative with respect to inputs. """  *# Unpack cache values*  x, w, z, out = cache  *# Use values in cache to compute derivatives*  dx = *# Derivative of loss with respect to x*  dw = *# Derivative of loss with respect to w*  **return** dx, dw |

We will also explore different update rules for optimization, and introduce Dropout as a regularizer and Batch Normalization as a tool to more efficiently optimize deep networks.

## 2.注意layer之间的forward和backward是怎么运行的



## 3.batch\_size/epoc/iteration的区别

epoch：所有的training data完成一次Forword运算以及一次BP运算.

Batch\_size:一次Forword运算以及BP运算中所需要的训练样本数目.

Iteration:每个iter都会随机取batch\_size这么多的数据来训练.

Iteration\*batch\_size=epoch\*num\_train

Iteration=epoch\*num\_train/batch\_size

所以solver里面train函数这么写:

|  |
| --- |
| def train(self):  """  Run optimization to train the model.  """  num\_train = self.X\_train.shape[0]  iterations\_per\_epoch = max(num\_train // self.batch\_size, 1)  num\_iterations = self.num\_epochs \* iterations\_per\_epoch  for t in range(num\_iterations):  self.\_step()#这里取batch\_size这么多的数据训练 |

## Optimazation

Momentum update:

v **=** mu **\*** v **-** learning\_rate **\*** dx *# integrate velocity*

x **+=** v *# integrate position*

RMSprop:

cache **=** decay\_rate **\*** cache **+** (1 **-** decay\_rate) **\*** dx**\*\***2

x **+=** **-** learning\_rate **\*** dx **/** (np**.**sqrt(cache) **+** eps)

Adam:

这里注意t要先+1

*# t is your iteration counter going from 1 to infinity*

m **=** beta1**\***m **+** (1**-**beta1)**\***dx

mt **=** m **/** (1**-**beta1**\*\***t)

v **=** beta2**\***v **+** (1**-**beta2)**\***(dx**\*\***2)

vt **=** v **/** (1**-**beta2**\*\***t)

x **+=** **-** learning\_rate **\*** mt **/** (np**.**sqrt(vt) **+** eps)

## Unknown word

Modular 模块化

Monolithic 整体的

Affine 仿射

# Assignment2\_Q2: Batch Normalization

Machine learning methods tend to work better when their input data consists of uncorrelated features with zero mean and unit variance.

为了达到这种目的,最简单的方式就是把input data去相关,这可以保证第一层网络看到一个很好的分布,但是在deeper的网络的激活值里,无法继续维持这种不相关性.

更糟糕的是,随着网络权值的更新,每一层feature 的分布也会改变.这会使得训练网络更加困难.为了克服这种问题,我们就引入了batch normalization 层.

Batch normalization做什么:在训练过程计算一个minibatch的标准差和均值,测试过程用来对feature进行中心化和归一化.

问题:可能会导致网络表征力下降.因为对于有一些层来说,非中心化,非归一化可能更好一些.

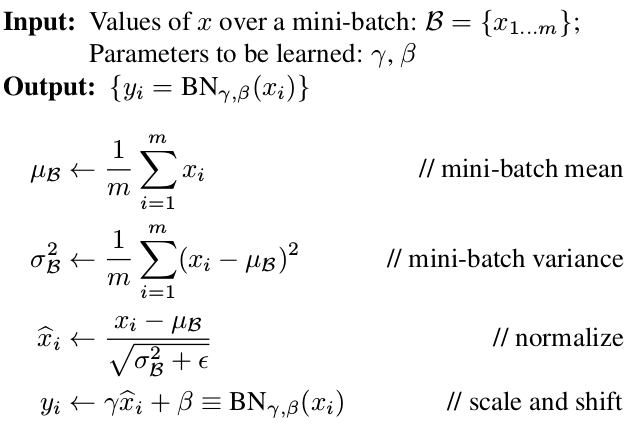
## 1.论文阅读Batch Normalization

Batch normalization主要是为了解决由于训练的过程中,参数(w)的更新,导致后面层input data的变化,这让我们必须使用较小的learning rate,并且谨慎地初始化w.Batch normalization用每次都把小batch里训练的样本进行归一化的方式,克服了以上的困难,有时候甚至不需要dropout.

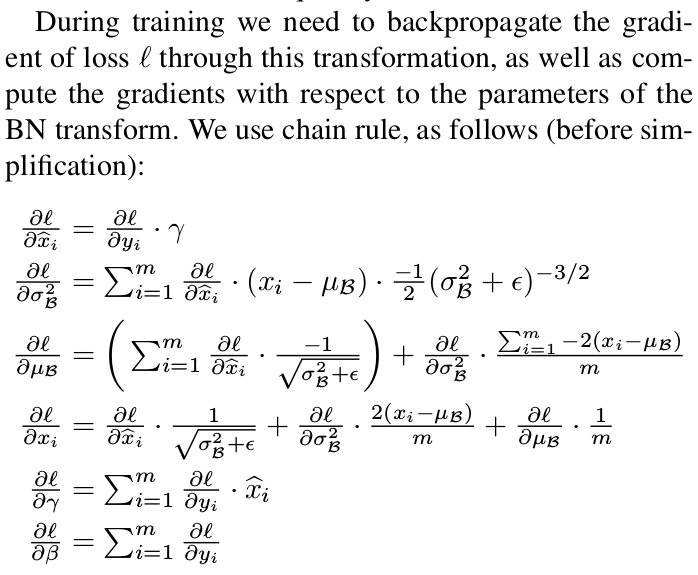
核心问题:

Fixed distribution of inputs to a sub-network would have positive consequences for the layers outside the sub-network, as well.

前向算法:



BP过程:



## 2.实现中的问题记录

### 2.1Batch Normalization: alternative backward

这里主要是让我们仿照sigmoid,不使用中间变量,直接用公式推导梯度.

参考: http://www.bubuko.com/infodetail-2031559.html

### 2.2把batch normalization加入FCN

Fully Connected Nets with Batch Normalization:

you should insert a batch normalization layer before each ReLU nonlinearity. The outputs from the last layer of the network should not be normalized.

这个cell里主要需要在initial,loss,gradient三个位置添加batch normalization的部分.

注意:

1.一定要判断是if self.use\_batchnorm and layer!=self.num\_layers-1

2.forward的时候bn层放在relu前面,bp的时候放在relu后面!

## 3.baseline,benchmark,sota的区别

Sota:state-of-the-art的缩写,表示目前最牛的算法.

Benchmark:一般是和同行中比较牛的算法比较，比牛算法还好，那你可以考虑发好一点的会议/期刊；

Baseline:比这个算法性能还差的基本上不能接受的.

baseline一般是自己算法优化和调参过程中自己和自己比较，目标是越来越好，当性能超过benchmark时，可以发表了，当性能甚至超过SOTA时，恭喜你，考虑投顶会顶刊啦。

## Unknown word

Straightforward 简单的

Decorrelate 去相关

Deviation 偏离

# Assignment2\_Q3: Dropout

1.生成mask的方法:

参考http://blog.csdn.net/hjimce/article/details/50413257中keras源码的实现.

参考numpy官方文档中生成二项分布数的教程:

<https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.random.binomial.html>

注意p应该是生成1的概率,所以要用(1-丢弃的概率)得到,这里n的含义是抛硬币的次数.

|  |
| --- |
| retain\_prob=1.0-p  mask = np.random.binomial(n=1, p=retain\_prob, size=x.shape)  out=x\*mask |

2.bp过程

|  |
| --- |
| dx=dout\*mask |

3.在网络结构中

Specificially, if the constructor the the net receives a nonzero value for the dropout parameter, then the net should add dropout immediately after every ReLU nonlinearity.

4.一些可能出现的bug

永远,不要,忘了,在求loss和gradient的时候先看一下要不要use\_dropout.

## Unknown word

Straightforward 简单的

Decorrelate 去相关

Deviation 偏离

# Video5.Lecture 5 | Convolutional Neural Networks

## Unknown word