个人资料

xqp_dream

访问： 11311次
积分： 206
等级： BLOG 2
排名： 千里之外

原创： 8篇 转载： 9篇
译文： 0篇 评论： 1条

文章搜索

文章分类

机器学习 (4)
c++ (2)
deep learning (6)
linux (3)

文章存档

2016年08月 (7)
2016年07月 (2)
2016年06月 (1)
2016年04月 (1)
2016年03月 (1)

展开

阅读排行

CNN卷积神经网络--反向 (1774)
CNN卷积神经网络--反向 (1617)
CNN卷积神经网络---反向 (1377)
stack around the variabl (1118)
CNN卷积神经网络--反向 (1064)
CNN卷积神经网络--反向 (1053)
RPCA图像处理中的矩阵 (929)
Linux makefile 教程 非常 (607)
在 linux 下使用 CMake 构 (278)
Caffe学习：Layers (263)

评论排行

Caffe学习：Layers (1)

【活动】2017 CSDN博客专栏评选 【5月书讯】流畅的Python，终于等到你！ CSDN日报20170519 ——《思维的局限》 Kotlin 专场

[置顶] CNN卷积神经网络--反向传播（3，Notes on Convolutional Neural Networks）

标签： deeplearning

2016-08-30 12:00 1786人阅读 评论((, , ,

分类： deep learning（5）

一、介绍

这个文档讨论的是CNNs的推导和实现。CNN架构的连接比权值要多很多，这实际上就隐含着实现了某种形式的规则化。这种特别的网络假定了我们希望通过数据驱动的方式学习到一些滤波器，作为提取输入的特征的一种方法。

本文中，我们先对训练全连接网络的经典BP算法做一个描述，然后推导2D CNN网络的卷积层和子采样层的BP权值更新方法。在推导过程中，我们更强调实现的效率，所以会给出一些Matlab代码。最后，我们转向讨论如何自动地学习组合前一层的特征maps，特别地，我们还学习特征maps的稀疏组合。

二、全连接的反向传播算法

典型的CNN中，开始几层都是卷积和下采样的交替，然后在最后一些层（靠近输出层的），都是全连接的一维网络。这时候我们已经将所有两维2D的特征maps转化为全连接的一维网络的输入。这样，当你准备好将最终的2D特征maps输入到1D网络中时，一个非常方便的方法就是把所有输出的特征maps连接成一个长的输入向量。然后我们回到BP算法的讨论。（更详细的基础推导可以参考UFLDL中“反向传导算法”）。

2.1、Feedforward Pass前向传播

在下面的推导中，我们采用平方误差代价函数。我们讨论的是多类问题，共c类，共N个训练样本。

$$E^N = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(t_k^n - y_k^n)^2.$$

这里t_k^n表示第n个样本对应的标签的第k维。y_k^n表示第n个样本对应的网络输出的第k个输出。对于多类问题，输出一般组织为“one-of-c”的形式，也就是只有该输入对应的类的输出节点输出为正，其他类的位或者节点为0或者负数，这个取决于你输出层的激活函数。sigmoid就是0，tanh就是-1.

因为在全部训练集上的误差只是每个训练样本的误差的总和，所以这里我们先考虑对于一个样本的BP。对于第n个样本的误差，表示为：

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(t_k^n - y_k^n)^2 = \frac{1}{2}\|\mathbf{t}^n - \mathbf{y}^n\|_2^2.$$

http://blog.csdn.net/zouxy09

传统的全连接神经网络中，我们需要根据BP规则计算代价函数E关于网络每一个权值的偏导数。我们用l来表示当前层，那么当前层的输出可以表示为：

$$\mathbf{x}^\ell = f(\mathbf{u}^\ell), \text{ with } \mathbf{u}^\ell = \mathbf{W}^\ell\mathbf{x}^{\ell-1} + \mathbf{b}^\ell$$

输出激活函数f(.)可以有很多种，一般是sigmoid函数或者双曲线正切函数。sigmoid将输出压缩到[0, 1]，所以最后的输出平均值一般趋于0 。所以如果将我们的训练数据归一化为零均值和方差为1，可以在梯度下降的过程中增加收敛性。对于归一化的数据集来说，双曲线正切函数也是不错的选择。

CNN卷积神经网络--反向 (0)
linux 下 .sh 文件语法 (0)
c++调用matlab编程 (0)
RPCA图像处理中的矩阵 (0)
奇异值分解SVD原理及其 (0)
PCA (0)

驱动小白和硬件老司机关于硬件那点事儿的一次密谈》
* 简单粗暴地入门机器学习
* AntShares区块链的节点部署与搭建私有链
* 分布式机器学习的集群方案介绍之HPC实现
* Android 音频系统：从AudioTrack 到 AudioFlinger

最新评论

Caffe学习：Layers
nzzfsw: 请问楼主的caffe包是从哪里下载的？我的caffe里缺少vision_layers.hpp等文件啊...

2.2、Backpropagation Pass反向传播

反向传播回来的误差可以看做是每个神经元的基的灵敏度sensitivities（灵敏度的意思就是我们的基b变化多少，误差会变化多少，也就是误差对基的变化率，也就是导数了），定义如下：（第二个等号是根据求导的链式法则得到的）

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial u}\frac{\partial u}{\partial b} = \delta$$

因为∂u/∂b=1，所以∂E/∂b=∂E/∂u=δ，也就是说bias基的灵敏度∂E/∂b=δ和误差E对一个节点全部输入u的导数∂E/∂u是相等的。这个导数就是让高层误差反向传播到底层的神来之笔。反向传播就是用下面这条关系式：（下面这条式子表达的就是第l层的灵敏度，就是）

$$\delta^{\ell} = (W^{\ell+1})^T \delta^{\ell+1} \circ f'(\mathbf{u}^{\ell})$$ 公式（1）

这里的“∘”表示每个元素相乘。输出层的神经元的灵敏度是不一样的：

$$\delta^{L} = f'(\mathbf{u}^{L}) \circ (\mathbf{y}^{n} - \mathbf{t}^{n}).$$

最后，对每个神经元运用delta（即δ）规则进行权值更新。具体来说就是，对一个给定的神经元，得到它的输入，然后用这个神经元的delta（即δ）来进行缩放。用向量的形式表述就是，对于第l层，误差对于该层每一个权值（组合为矩阵）的导数是该层的输入（等于上一层的输出）与该层的灵敏度（该层每个神经元的δ组合成一个向量的形式）的叉乘。然后得到的偏导数乘学习率就是该层的神经元的权值的更新了：

$$\frac{\partial E}{\partial \mathbf{W}^{\ell}} = \mathbf{x}^{\ell-1}(\boldsymbol{\delta}^{\ell})^T$$
$$\Delta \mathbf{W}^{\ell} = -\eta \frac{\partial E}{\partial \mathbf{W}^{\ell}}$$ 公式（2）

对于bias基的更新表达式差不多。实际上，对于每一个权值$(W)_{ij}$都有一个特定的学习率η_{ij}。

三、Convolutional Neural Networks 卷积神经网络

3.1、Convolution Layers 卷积层

我们现在关注网络中卷积层的BP更新。在一个卷积层，上一层的特征maps被一个可学习的卷积核进行卷积，然后通过一个激活函数，就可以得到输出特征map。每一个输出map可能是组合卷积多个输入maps的值：

$$\mathbf{x}_j^{\ell} = f\left(\sum_{i \in M_j} \mathbf{x}_i^{\ell-1} * \mathbf{k}_{ij}^{\ell} + b_j^{\ell}\right)$$

这里M_j表示选择的输入maps的集合，那么到底选择哪些输入maps呢？有选择一对的或者三个的。但下面我们会讨论如何去自动选择需要组合的特征maps。每一个输出map会给一个额外的偏置b，但是对于一个特定的输出map，卷积每个输入maps的卷积核是不一样的。也就是说，如果输出特征map j和输出特征map k都是从输入map i中卷积求和得到，那么对应的卷积核是不一样的。

3.1.1、Computing the Gradients梯度计算

我们假定每个卷积层l都会接一个下采样层l+1。对于BP来说，根据上文我们知道，要想求得层l的每个神经元对应的权值的权值更新，就需要先求层l的每一个神经节点的灵敏度δ（也就是权值更新的公式（2））。为了求这个灵敏度我们就需要先对下一层的节点（连接到当前层l的感兴趣节点的第l+1层的节点）的灵敏度求和（得到δ^{l+1}），然后乘以这些连接对应的权值（连接第l层感兴趣节点和第l+1层节点的权值）W。再乘以当前层l的该神经元节点的输入u的激活函数f的导数值（也就是那个灵敏度反向传播的公式（1）的δ^{l}的求解），这样就可以得到当前层l每个神经节点对应的灵敏度δ^{l}了。

然而，因为下采样的存在，采样层的一个像素（神经元节点）对应的灵敏度δ对应于卷积层（上一层）的输出map的一块像素（采样窗口大小）。因此，层l中的一个map的每个节点只与l+1层中相应map的一个节点连接。

为了有效计算层l的灵敏度，我们需要上采样upsample 这个下采样downsample层对应的灵敏度map（特征map中每个像素对应一个灵敏度，所以也组成一个map），这样才使得这个灵敏度map大小与卷积层的map大小一致，然后再将层l的map的激活值的偏导数与从第l+1层的上采样得到的灵敏度map逐元素相乘（也就是公式（1））。

在下采样层map的权值都取一个相同值β，而且是一个常数。所以我们只需要将上一个步骤得到的结果乘以一个β就可以完成第l层灵敏度δ的计算。

我们可以对卷积层中每一个特征map j重复相同的计算过程。但很明显需要匹配相应的子采样层的map（参考公式（1））：

$$\delta_j^\ell = \beta_j^{\ell+1}\left(f'(\mathbf{u}_j^\ell) \circ \mathrm{up}(\delta_j^{\ell+1})\right)$$

up(.)表示一个上采样操作。如果下采样的采样因子是n的话，它简单的将每个像素水平和垂直方向上拷贝n次。这样就可以恢复原来的大小了。实际上，这个函数可以用Kronecker乘积来实现：

$$\mathrm{up}(\mathbf{x}) \equiv \mathbf{x} \otimes \mathbf{1}_{n\times n}.$$

好，到这里，对于一个给定的map，我们就可以计算得到其灵敏度map了。然后我们就可以通过简单的对层l中的灵敏度map中所有节点进行求和快速的计算bias基的梯度了：

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^\ell)_{uv}.$$

公式（3）

最后，对卷积核的权值的梯度就可以用BP算法来计算了（公式（2））。另外，很多连接的权值是共享的，因此，对于一个给定的权值，我们需要对所有与该权值有联系（权值共享的连接）的连接对该点求梯度，然后对这些梯度进行求和，就像上面对bias基的梯度计算一样：

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^\ell} = \sum_{u,v} (\delta_j^\ell)_{uv} (\mathbf{p}_i^{\ell-1})_{uv}$$

这里，$(\mathbf{p}_i^{\ell-1})_{uv}$是$\mathbf{x}_i^{\ell-1}$中的在卷积的时候与$\mathbf{k}_{ij}^\ell$逐元素相乘的patch，输出卷积map的(u, v)位置的值是由上一层的(u, v)位置的patch与卷积核k_ij逐元素相乘的结果。

咋一看，好像我们需要煞费苦心地记住输出map（和对应的灵敏度map）每个像素对应于输入map的哪个patch。但实际上，在Matlab中，可以通过一个代码就实现。对于上面的公式，可以用Matlab的卷积函数来实现：

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^\ell} = \mathrm{rot180}(\mathrm{conv2}(\mathbf{x}_i^{\ell-1}, \mathrm{rot180}(\delta_j^\ell), \text{'valid'}))$$

我们先对delta灵敏度map进行旋转，这样就可以进行互相关计算，而不是卷积（在卷积的数学定义中，特征矩阵（卷积核）在传递给conv2时需要先翻转（flipped）一下。也就是颠倒下特征矩阵的行和列）。然后把输出反旋转回来，这样我们在前向传播进行卷积的时候，卷积核才是我们想要的方向。

3.2、Sub-sampling Layers 子采样层

对于子采样层来说，有N个输入maps，就有N个输出maps，只是每个输出map都变小了。

$$\mathbf{x}_j^\ell = f\left(\beta_j^\ell \mathrm{down}(\mathbf{x}_j^{\ell-1}) + b_j^\ell\right)$$

down(.)表示一个下采样函数。典型的操作一般是对输入图像的不同nxn的块的所有像素进行求和。这样输出图像在两个维度上都缩小了n倍。每个输出map都对应一个属于自己的乘性偏置β和一个加性偏置b。

3.2.1、Computing the Gradients 梯度计算

这里最困难的是计算灵敏度map。一旦我们得到这个了，那我们唯一需要更新的偏置参数β和b就可以轻而易举了（公式（3））。如果下一个卷积层与这个子采样层是全连接的，那么就可以通过BP来计算子采样层的灵敏度maps。

我们需要计算卷积核的梯度，所以我们必须找到输入map中哪个patch对应输出map的哪个像素。这里，就是必须找到当前层的灵敏度map中哪个patch对应与下一层的灵敏度map的给定像素，这样才可以利用公式（1）那样的δ递推，也就是灵敏度反向传播回来。另外，需要乘以输入patch与输出像素之间连接的权值，这个权值实际上就是卷积核的权值（已旋转的）。

$$\delta_j^\ell = f'(\mathbf{u}_j^\ell) \circ \mathrm{conv2}(\delta_j^{\ell+1}, \mathrm{rot180}(\mathbf{k}_j^{\ell+1}), \text{'full'})$$

在这之前，我们需要先将核旋转一下，让卷积函数可以实施互相关计算。另外，我们需要对卷积边界进行处理，但在Matlab里面，就比较容易处理。Matlab中全卷积会对缺少的输入像素补0。

到这里，我们就可以对b和β计算梯度了。首先，加性基b的计算和上面卷积层的一样，对灵敏度map中所有元素加起来就可以了：

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^\ell)_{uv}.$$

而对于乘性偏置β，因为涉及到了在前向传播过程中下采样map的计算，所以我们最好在前向的过程中保存好这些maps，这样在反向的计算中就不用重新计算了。我们定义：

$$\mathbf{d}_j^\ell = \mathrm{down}(\mathbf{x}_j^{\ell-1}).$$

这样，对β的梯度就可以用下面的方式计算：

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^\ell \circ \mathbf{d}_j^\ell)_{uv}.$$

3.3、Learning Combinations of Feature Maps 学习特征map的组合

大部分时候，通过卷积多个输入maps，然后再对这些卷积值求和得到一个输出map，这样的效果往往是比较好的。在一些文献中，一般是人工选择哪些输入maps去组合得到一个输出map。但我们这里尝试去让CNN在训练的过程中学习这些组合，也就是让网络自己学习挑选哪些输入maps来计算得到输出map才是最好的。我们用αij表示在得到第j个输出map的其中第i个输入map的权值或者贡献。这样，第j个输出map可以表示为：

$$\mathbf{x}_j^\ell = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell) + b_j^\ell\right)$$

http://blog.csdn.net/zouxy09

需要满足约束：

$$\sum_i \alpha_{ij} = 1, \quad \text{and} \quad 0 \le \alpha_{ij} \le 1.$$

http://blog.csdn.net/zouxy09

这些对变量αij的约束可以通过将变量αij表示为一个组无约束的隐含权值c_{ij}的softmax函数来加强。（因为softmax的因变量是自变量的指数函数，他们的变化率会不同）。

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_k \exp(c_{kj})}.$$

因为对于一个固定的j来说，每组权值c_{ij}都是和其他组的权值独立的，所以为了方面描述，我们把下标j去掉，只考虑一个map的更新，其他map的更新是一样的过程，只是map的索引j不同而已。

Softmax函数的导数表示为：

$$\frac{\partial \alpha_k}{\partial c_i} = \delta_{ki}\alpha_i - \alpha_i\alpha_k$$

这里的δ是Kronecker delta。对于误差对于第l层变量αi的导数为：

$$\frac{\partial E}{\partial \alpha_i} = \frac{\partial E}{\partial u^\ell}\frac{\partial u^\ell}{\partial \alpha_i} = \sum_{u,v}\left(\boldsymbol{\delta}^\ell \circ (\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell)\right)_{uv}.$$

http://blog.csdn.net/zouxy09

最后就可以通过链式规则去求得代价函数关于权值c_i的偏导数了：

$$\begin{aligned}\frac{\partial E}{\partial c_i} &= \sum_k \frac{\partial E}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} \\ &= \alpha_i\left(\frac{\partial E}{\partial \alpha_i} - \sum_k \frac{\partial E}{\partial \alpha_k}\alpha_k\right).\end{aligned}$$

http://blog.csdn.net/zouxy09

3.3.1、Enforcing Sparse Combinations 加强稀疏性组合

为了限制αi是稀疏的，也就是限制一个输出map只与某些而不是全部的输入maps相连。我们在整体代价函数里增加稀疏约束项Ω(α)。对于单个样本，重写代价函数为：

$$\tilde{E}^n = E^n + \lambda \sum_{i,j} |(\boldsymbol{\alpha})_{ij}|$$

然后寻找这个规则化约束项对权值ci求导的贡献。规则化项Ω(α)对αi求导是：

$$\frac{\partial \Omega}{\partial \alpha_i} = \lambda \operatorname{sign}(\alpha_i)$$

然后，通过链式法则，对ci的求导是：

$$\begin{aligned}\frac{\partial \Omega}{\partial c_i} &= \sum_k \frac{\partial \Omega}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} \\ &= \lambda\left(|\alpha_i| - \alpha_i \sum_k |\alpha_k|\right).\end{aligned}$$

http://blog.csdn.net/zouxy09

所以，权值ci最后的梯度是：

$$\frac{\partial \tilde{E}^n}{\partial c_i} = \frac{\partial E^n}{\partial c_i} + \frac{\partial \Omega}{\partial c_i}.$$

3.4、Making it Fast with MATLAB

CNN的训练主要是在卷积层和子采样层的交互上，其主要的计算瓶颈是：

1）前向传播过程：下采样每个卷积层的maps；

2）反向传播过程：上采样高层子采样层的灵敏度map，以匹配底层的卷积层输出maps的大小；

3）sigmoid的运用和求导。

对于第一和第二个问题，我们考虑的是如何用Matlab内置的图像处理函数去实现上采样和下采样的操作。对于上采样，imresize函数可以搞定，但需要很大的开销。一个比较快速的版本是使用Kronecker乘积函数kron。通过一个全一矩阵ones来和我们需要上采样的矩阵进行Kronecker乘积，就可以实现上采样的效果。对于前向传播过程中的下采样，imresize并没有提供在缩小图像的过程中还计算nxn块内像素的和的功能，所以没法用。一个比较好和快速的方法是用一个全一的卷积核来卷积图像，然后简单的通过标准的索引方法来采样最后卷积结果。例如，如果下采样的域是2x2的，那么我们可以用2x2的元素全是1的卷积核来卷积图像。然后再卷积后的图像中，我们每个2个点采集一次数据，y=x(1:2:end,1:2:end)，这样就可以得到了两倍下采样，同时执行求和的效果。

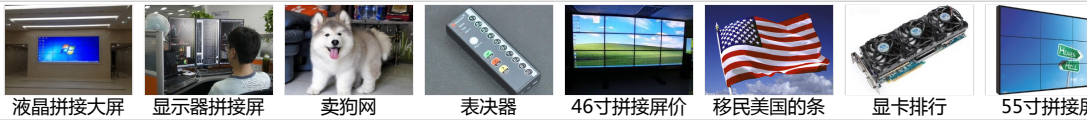
对于第三个问题，实际上有些人以为Matlab中对sigmoid函数进行inline的定义会更快，其实不然，Matlab与C/C++等等语言不一样，Matlab的inline反而比普通的函数定义更非时间。所以，我们可以直接在代码中使用计算sigmoid函数及其导数的真实代码。

顶 1　踩 0

上一篇　CNN卷积神经网络--反向传播（2，前向传播）
下一篇　CNN卷积神经网络--反向传播（4，代码理解）

相关文章推荐

- CNN卷积神经网络反向传播机制的理解
- CNN卷积神经网络---反向传播1全链接bp算法
- 斯坦福CS231n - CNN for Visual Recognition4-lectu…
- 卷积神经网络Convolutional Neural Networks
- 卷积神经网络小结Convolutional Neural Networks
- Convolutional Neural Networks 卷积神经网络
- 卷积神经网络Convolutional Neural Networks
- 深度学习FPGA实现基础知识10Deep Learning深度…
- 深度卷积神经网络学习笔记2步长不为1的卷积前向传…
- 卷积神经网络CNN反向传播算法



猜你在找

《C语言/C++学习指南》加密解密篇（安全相关算法）
C语言系列之 递归算法示例与 Windows 趣味小项目
C语言系列之 字符串相关算法
C语言系列之 字符串压缩算法与结构体初探
C语言系列之 快速排序与全排列算法
C语言系列之 数组与算法实战
使用决策树算法对测试数据进行分类实战
使用决策树算法对测试数据进行分类实战
模板匹配的字符识别(OCR)算法原理
数据结构与算法在实战项目中的应用

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN
Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5

Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 | 江苏乐知网络技术有限公司
京 1999-2017, CSDN.NET, All Rights Reserved