个人资料

lingerlanlan

caffe源码分析--poolinger_layer.cpp

分类： 深度学习（deep learning） 2014-07-30 19:46 2635人阅读 评论(6) 收藏 举报

caffe deep learning 机器学习 源码分析 神经网络

码分析--poolinger_layer.cpp

对于采样层，cafffe里实 l平均采样的算法。

最大采样，给定一个扫描窗口，找最大值，

平均采样，扫描窗口内所有值的平均值。

其实对于caffe的实现一直有个疑问，

就是每一层貌似没有绑定一个激活函数？

看ufldl教程，感觉激活函数是必要存在的。

这怎么解释呢？

看到源码中，看到一些激活函数，比如sigmoid_layer.cpp和sigmoid_layer.cu。

也就是说，激活函数作为layer层面来实现了。当然，还有tanh_layer和relu_layer。

那，这个意思是说，让我们建立网络的时候更加随意，可自由搭配激活函数吗？

但是，我看了caffe自带的那些例子，貌似很少见到用了激活函数layer的，顶多看到用了relu_layer，其他的没见过。

这意思是说，激活函数不重要吗？真是费解啊。

```cpp
// Copyright 2013 Yangqing Jia

#include <algorithm>
#include <cfloat>
#include <vector>

#include "caffe/layer.hpp"
#include "caffe/vision_layers.hpp"
#include "caffe/util/math_functions.hpp"

using std::max;
using std::min;
```

2015年07月 (3)
2015年06月 (3)
2015年05月 (3)
2015年04月 (8)
展开

最新评论

总结一下用caffe跑图片数据的研
liangzhituzi: @zzq1989_:可能是那两个文件路径的问题，可以看看train_prototxt里面的路径

deep learning实践经验总结
查志强: 问下，怎样判断“错误”的标签？

神经网络：caffe特征可视化的代
fqss0436: 博主，您好，谢谢您分享代码。在调试您的代码时，程序中断于175行caffe_test_net.For...

我所写的CNN框架 VS caffe
gzp95: 楼主，求问一下您写的代码的速度和caffe的速度有多大的差距。因为最近在实现word2vec的cud...

总结一下用caffe跑图片数据的研
依然_范佩西11: 训练完的模型，如是调用呢，能说下能么测试单张图像或者批量图像的流程么

Dota全图那些事儿
女主、女主: 。。。单机理论效果，实际不好用啊。。。。。支持一下~不错的

caffe源码修改：抽取任意一张图
wwdzhtxnjwcnmd: 想请教一下博主，caffe网络中batch_size和crop_size这两个参数的含义是什么？哪一...

caffe源码分析--data_layer.cpp
沧海1梦: 请问caffe中如何修改输入和裁剪尺寸，因为我的图像大小是48的，想通过修改alexnet来训练，还...

caffe卷积神经网络框架安装
yang123jx: 我也遇到relu_layer.cu:29 check failed error == cudaSuc...

caffe卷积神经网络框架安装
yang123jx: 我也遇到relu_layer.cu:29 check failed error == cudaSuc...

阅读排行

总结一下用caffe跑图片数 (7192)
word2vector学习笔记（ (6942)
caffe神经网络框架的辅助 (6147)
caffe源码修改：抽取任意 (5905)
caffe卷积神经网络框架安 (5550)
caffe源码分析--data_lay (5374)
神经网络：caffe特征可视 (4679)
word2vec源码解析之wo (4510)
caffe源码分析--Blob类代 (4386)
deep learning实践经验总 (4225)

推荐文章

```cpp
namespace caffe {

template <typename Dtype>
void PoolingLayer<Dtype>::SetUp(const vector<Blob<Dtype>*>& bottom,
      vector<Blob<Dtype>*>* top) {
  CHECK_EQ(bottom.size(), 1) << "PoolingLayer takes a single blob as input.";
  CHECK_EQ(top->size(), 1) << "PoolingLayer takes a single blob as output.";
  KSIZE_ = this->layer_param_.kernelsize();//核大小
  STRIDE_ = this->layer_param_.stride();//步长
  CHANNELS_ = bottom[0]->channels();//通道
  HEIGHT_ = bottom[0]->height();//高
  WIDTH_ = bottom[0]->width();//宽
  POOLED_HEIGHT_ = static_cast<int>(
      ceil(static_cast<float>(HEIGHT_ - KSIZE_) / STRIDE_)) + 1;//计算采样之后的高
  POOLED_WIDTH_ = static_cast<int>(
      ceil(static_cast<float>(WIDTH_ - KSIZE_) / STRIDE_)) + 1;//计算采样之后的宽
  (*top)[0]->Reshape(bottom[0]->num(), CHANNELS_, POOLED_HEIGHT_,//采样之后大小
      POOLED_WIDTH_);
  // If stochastic pooling, we will initialize the random index part.
  if (this->layer_param_.pool() == LayerParameter_PoolMethod_STOCHASTIC) {
    rand_idx_.Reshape(bottom[0]->num(), CHANNELS_, POOLED_HEIGHT_,
      POOLED_WIDTH_);
  }
}

// TODO(Yangqing): Is there a faster way to do pooling in the channel-first
// case?
template <typename Dtype>
void PoolingLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
      vector<Blob<Dtype>*>* top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();//采样层输入
  Dtype* top_data = (*top)[0]->mutable_cpu_data();//采样层输出
  // Different pooling methods. We explicitly do the switch outside the for
  // loop to save time, although this results in more codes.
  int top_count = (*top)[0]->count();
  switch (this->layer_param_.pool()) {
  case LayerParameter_PoolMethod_MAX://最大采样方法
    // Initialize
    for (int i = 0; i < top_count; ++i) {
      top_data[i] = -FLT_MAX;
    }
    // The main loop
    for (int n = 0; n < bottom[0]->num(); ++n) {
      for (int c = 0; c < CHANNELS_; ++c) {
        for (int ph = 0; ph < POOLED_HEIGHT_; ++ph) {
          for (int pw = 0; pw < POOLED_WIDTH_; ++pw) {
            int hstart = ph * STRIDE_;
            int wstart = pw * STRIDE_;
            int hend = min(hstart + KSIZE_, HEIGHT_);
            int wend = min(wstart + KSIZE_, WIDTH_);
            for (int h = hstart; h < hend; ++h) {//找出核范围内最大
              for (int w = wstart; w < wend; ++w) {
                top_data[ph * POOLED_WIDTH_ + pw] =
                  max(top_data[ph * POOLED_WIDTH_ + pw],
                      bottom_data[h * WIDTH_ + w]);
              }
            }
          }
        }
        // compute offset 指针移动到下一个channel。注意代码这里的位置。采样是针对每个channel的。
        bottom_data += bottom[0]->offset(0, 1);
        top_data += (*top)[0]->offset(0, 1);
      }
    }
    break;
  case LayerParameter_PoolMethod_AVE:
    for (int i = 0; i < top_count; ++i) {
      top_data[i] = 0;
    }
    // The main loop
    for (int n = 0; n < bottom[0]->num(); ++n) {
      for (int c = 0; c < CHANNELS_; ++c) {
        for (int ph = 0; ph < POOLED_HEIGHT_; ++ph) {
          for (int pw = 0; pw < POOLED_WIDTH_; ++pw) {
            int hstart = ph * STRIDE_;
            int wstart = pw * STRIDE_;
            int hend = min(hstart + KSIZE_, HEIGHT_);
            int wend = min(wstart + KSIZE_, WIDTH_);
```

```
            for (int h = hstart; h < hend; ++h) {//核范围内算平均
              for (int w = wstart; w < wend; ++w) {
                top_data[ph * POOLED_WIDTH_ + pw] +=
                    bottom_data[h * WIDTH_ + w];
              }
            }
            top_data[ph * POOLED_WIDTH_ + pw] /=
                (hend - hstart) * (wend - wstart);
          }
        }
        // compute offset
        bottom_data += bottom[0]->offset(0, 1);
        top_data += (*top)[0]->offset(0, 1);
      }
    }
    break;
  case LayerParameter_PoolMethod_STOCHASTIC://这种算法这里未实现
    NOT_IMPLEMENTED;
    break;
  default:
    LOG(FATAL) << "Unknown pooling method.";
  }
}

template <typename Dtype>
Dtype PoolingLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
      const bool propagate_down, vector<Blob<Dtype>*>* bottom) {
  if (!propagate_down) {
    return Dtype(0.);
  }
  const Dtype* top_diff = top[0]->cpu_diff();
  const Dtype* top_data = top[0]->cpu_data();
  const Dtype* bottom_data = (*bottom)[0]->cpu_data();
  Dtype* bottom_diff = (*bottom)[0]->mutable_cpu_diff();
  // Different pooling methods. We explicitly do the switch outside the for
  // loop to save time, although this results in more codes.
  memset(bottom_diff, 0, (*bottom)[0]->count() * sizeof(Dtype));
  switch (this->layer_param_.pool()) {
  case LayerParameter_PoolMethod_MAX:
    // The main loop
    for (int n = 0; n < top[0]->num(); ++n) {
      for (int c = 0; c < CHANNELS_; ++c) {
        for (int ph = 0; ph < POOLED_HEIGHT_; ++ph) {
          for (int pw = 0; pw < POOLED_WIDTH_; ++pw) {
            int hstart = ph * STRIDE_;
            int wstart = pw * STRIDE_;
            int hend = min(hstart + KSIZE_, HEIGHT_);
            int wend = min(wstart + KSIZE_, WIDTH_);
            for (int h = hstart; h < hend; ++h) {
              for (int w = wstart; w < wend; ++w) {
                bottom_diff[h * WIDTH_ + w] +=//采样层输出的残传播给输入。由于是最大采样方法，输出存的都是输入范围内最大的值，所以残差传播的时候也只有范围内最大的值受影响
                    top_diff[ph * POOLED_WIDTH_ + pw] *
                    (bottom_data[h * WIDTH_ + w] ==
                        top_data[ph * POOLED_WIDTH_ + pw]);
              }
            }
          }
        }
        // offset  移动到下一个channel
        bottom_data += (*bottom)[0]->offset(0, 1);
        top_data += top[0]->offset(0, 1);
        bottom_diff += (*bottom)[0]->offset(0, 1);
        top_diff += top[0]->offset(0, 1);
      }
    }
    break;
  case LayerParameter_PoolMethod_AVE:
    // The main loop
    for (int n = 0; n < top[0]->num(); ++n) {
      for (int c = 0; c < CHANNELS_; ++c) {
        for (int ph = 0; ph < POOLED_HEIGHT_; ++ph) {
          for (int pw = 0; pw < POOLED_WIDTH_; ++pw) {
            int hstart = ph * STRIDE_;
            int wstart = pw * STRIDE_;
            int hend = min(hstart + KSIZE_, HEIGHT_);
            int wend = min(wstart + KSIZE_, WIDTH_);
            int poolsize = (hend - hstart) * (wend - wstart);
            for (int h = hstart; h < hend; ++h) {
```

```
170.              for (int w = wstart; w < wend; ++w) {
171.                bottom_diff[h * WIDTH_ + w] +=//采样层输出的残差传播给输入，由于是平均采样，所以权重
    都是1 / poolsize。
172.                    top_diff[ph * POOLED_WIDTH_ + pw] / poolsize;
173.                }
174.              }
175.            }
176.          }
177.          // offset
178.          bottom_data += (*bottom)[0]->offset(0, 1);
179.          top_data += top[0]->offset(0, 1);
180.          bottom_diff += (*bottom)[0]->offset(0, 1);
181.          top_diff += top[0]->offset(0, 1);
182.        }
183.      }
184.      break;
185.    case LayerParameter_PoolMethod_STOCHASTIC:
186.      NOT_IMPLEMENTED;
187.      break;
188.    default:
189.      LOG(FATAL) << "Unknown pooling method.";
190.    }
191.    return Dtype(0.);
192.  }
193.
194.
195.  INSTANTIATE_CLASS(PoolingLayer);
196.
197.
198.  }  // namespace caffe
```

本文作者：linger

本文链接：http://blog.csdn.net/lingerlanlan/article/details/38294169

上一篇 word2vec源码解析之word2vec.c

下一篇 bag-of-words model的java实现

查看评论

3楼 visionlixu 2015-06-04 17:32发表

楼主，您好
我刚刚学习caffe,看了一部分caffe code。 发现求了图像均值之后，然后每张图片都减去图像均值。但我找不到哪里减去图像均值的代码。
希望楼主帮我指出来。谢谢

2楼 ok272733314 2015-05-03 18:19发表

博主您好，看了你几篇博客感觉你真的很厉害，想请教你几个问题，望不吝赐教~
1. 我现在在用caffe的python API 来完成一些简单的分类，就像caffe官网里提供的例子。但是我最近发现还是需要动手去改动一下c++的源码。当我开始着手c++ 工程的时候还是一头污水不知从何搞起。想问一下你用的什么IDE？怎么将工程导入到IDE里然后build的？会不会用到cmakelist.txt文件？该如何用呢？
2. tools 文件夹里面的cpp文件又该怎么样才能运行呢？我对这方面不是很了解，非常感谢

1楼 CH 2014-08-04 19:09发表

确实，搭建网络时可自由的为每一层搭配激活函数，各种非线性映射函数都是在caffe中都是原位运算，不再另行分配内存空间。目前都采用relu，可能是论文中说relu相比sigmod或tanh，能够加速网络参数的收敛，所以大家都这么用了~~

Re: lingerlanlan 2014-08-04 23:48发表

回复CH：原来如此啊！哪篇论文有提到这点？

Re: CH 2014-08-05 16:53发表

回复lingerlanlan：我目前看的文章中，有三篇提到了，列举如下，供您参考一下：1.《Deep Sparse Rectifier Neural Networks》2.《Improving neural networks by preventing co-adaptation of feature detectors》 的附录F33.《ImageNet Classification with Deep Convolutional Neural Networks》的3.1节。

Re: lingerlanlan 2014-08-05 21:51发表

回复CH：感谢！

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