Caffe源码解析3：Layer

转载请注明出处，楼燚(yì)航的blog，http://home.cnblogs.com/louyihang-loves-baiyan/

layer这个类可以说是里面最终的一个基本类了，深度网络呢就是一层一层的layer，相互之间通过blob传输数据连接起来。首先layer必须要实现一个forward function，前递函数当然功能可以自己定义啦，在forward中呢他会从input也就是Layer的bottom，对了caffe里面网络的前一层是叫bottom的，从bottom中获取blob，并且计算输出的Blob，当然他们也会实现一个反向传播，根据他们的input的blob以及output blob的error gradient 梯度误差计算得到该层的梯度误差。从公式中也可以看到：

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \sigma'(z^l)$$

首先来看layer类的构造部分，以及Public部分的函数

```
template <typename Dtype>
class Layer {
 public:
  explicit Layer(const LayerParameter& param)
    : layer_param_(param), is_shared_(false) {
      // Set phase and copy blobs (if there are any).
      phase_ = param.phase();
      if (layer_param_.blobs_size() > 0) {
        blobs_.resize(layer_param_.blobs_size());
        for (int i = 0; i < layer_param_.blobs_size(); ++i) {
          blobs_[i].reset(new Blob<Dtype>());
          blobs_[i]->FromProto(layer_param_.blobs(i));
        }
      }
    }
  virtual ~Layer() {}
```

首先获得当前网络的Phase，是train还是test，在初始化列表初始化LayerParameter,之后blobs_这里存放的是一个指向blob类的shared_ptr指针的一个vector，在这里是申请空间，然后将传入的layer_param中的blob拷贝过来。

```
void SetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
    InitMutex();
    CheckBlobCounts(bottom, top);
    LayerSetUp(bottom, top);
    Reshape(bottom, top);
    SetLossWeights(top);
  }
```

这里是Setup函数，首先check 这个bottom和top的blob是否正确，再调用Layersetup对每一具体的层做进一步设置，之后再做reshape来设置top blobs和internal buffer。最后再设置loss weight multiplier 的blob对每一个非零的loss和weight，一般这个方法被继承之后是不会被重写的。

```
virtual void LayerSetUp(const vector<Blob<Dtype>*>& bottom,const vector<Blob<Dtype>*>& top)
virtual inline bool ShareInParallel()
inline bool IsShared() const
inline void SetShared(bool is_shared)
```

LayerSetup就是对具体某一个layer的setup,被上面的那个函数所调用，ShareInParallel和IsShared和SetShared分别是用来返回并行状态和获得这一layer是否被多个nets所共享，默认是除了data layer都是关闭的。在多个GPU下的Train阶段以及share是true的情况下,is_shared将会被置成true。

```
virtual void Reshape(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) = 0;
```

这个reshape主要是layer用来根据输入的blob调节Internal buffer以及输出的Blob的

注意

接下来是几个最重要的函数，首先是**Forward.**这其实是一个装饰器，继承之后在调用的调用其相应的forward_cpu或者forward_gpu，根据输入的input data blob计算相应的output data blob，同时会反应这一层layer的total loss.

```
inline Dtype Forward(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top);
```

这里是**BackWard**，实现的是反向传播，也就是给定top blob额error gradient 计算得到bottom的error gradient。其输入时 output blobs ，在Ouput blobs里面的diff存储的就是其相应的error gradients。其中propagate_down这个参数跟Bottom的长度是一样的，每一个Index用来指定是否需要反向传播error gradients到对应的bottom blob。而bottom 这里面的diff 区域存放的就是BackWard计算出来相应的gradient error.

```
inline void Backward(const vector<Blob<Dtype>*>& top,
      const vector<bool>& propagate_down,
      const vector<Blob<Dtype>*>& bottom);
```

如果自己你要实现一个Layer的话，那么Forward_cpu和Backward_cpu 以及gpu（可选），应该要有自己的实现。

杰下来几个函数比较简单，统一做说明

```
vector<shared_ptr<Blob<Dtype> > >& blobs()\\返回blobs
const LayerParameter& layer_param() \\返回layer 的参数parameter
virtual void ToProto(LayerParameter* param, bool write_diff = false)\\将层参数写到Protobuffer里
inline Dtype loss(const int top_index) const \\给定index返回相应的scalar loss
inline void set_loss(const int top_index, const Dtype value)\\给定Index设置loss
virtual inline const char* type()\\返回layer的type
```

一下几个函数主要获得bottom或者top blob的数量状态，比较简单，看名字即可

```
virtual inline int ExactNumBottomBlobs()
virtual inline int MinBottomBlobs()
virtual inline int MaxBottomBlobs()
virtual inline int ExactNumTopBlobs()
virtual inline int MinTopBlobs()
virtual inline int MaxTopBlobs()
virtual inline bool EqualNumBottomTopBlobs()
virtual inline bool AutoTopBlobs()
```

AllowforceBackward用来设置是否强制梯度返回，因为有些层其实不需要梯度信息 ，后面两个函数分别查看以及设置是是否需要计算梯度。

```
virtual inline bool AllowForceBackward(const int bottom_index)
inline bool param_propagate_down(const int param_id)
inline void set_param_propagate_down(const int param_id, const bool value)
```

好，我们再后下面看，几个变量和函数都是保护变量

```
LayerParameter layer_param_; \\保存layer的参数 parameter
Phase phase_; \\标定阶段是train还是test
vector<shared_ptr<Blob<Dtype> > > blobs_; \\是Blob的一个集合，保存了learnbale参数
vector<bool> param_propagate_down_;\\标志位是否要计算param blob的梯度
vector<Dtype> loss_;\\用来表明那个top blob 有非零的权重
```

下面这几个函数，分别是计算cpu和gpu模式下的正向和反向传播

```
virtual void Forward_cpu(const vector<Blob<Dtype>*>& bottom, const vector<Blob<Dtype>*>& top)
virtual void Forward_gpu(const vector<Blob<Dtype>*>& bottom, const vector<Blob<Dtype>*>& top)
virtual void Backward_cpu(const vector<Blob<Dtype>*>& top,const vector<bool>&
propagate_down,const vector<Blob<Dtype>*>& bottom) = 0;
virtual void Backward_gpu(const vector<Blob<Dtype>*>& top,const vector<bool>&
propagate_down,const vector<Blob<Dtype>*>& bottom)
```

这个函数被setup调用，主要是check bottom和top 的blob是否match，这里面用了上面提到的ExactBottomBlobs（）等函数

```
virtual void CheckBlobCounts(const vector<Blob<Dtype>*>& bottom, const vector<Blob<Dtype>*>&
top)
```

SetLoss是非常重要的一个步骤，是被SetUp调用来初始化top bottom的weights，并且存储非零的loss weights在diff blob里面

```
inline void SetLossWeights(const vector<Blob<Dtype>*>& top)
```

私有变量和函数如下，东西比较少，主要是对并行中的锁进行控制

--gaosi123

2. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

BTW, 我电脑是i7 4790k + 16GB内存，所以硬件设备应该不会是限制。不知道问题出在哪里

--gaosi123

3. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

如果可以，欢迎留个email

--gaosi123

4. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

你好，我也是直接把DPM代码拷贝到工程里，但是想你这样直接拷进去不会报错吗？我直接拷贝进去按照你的来，报错信息如下：Error 4 error C2039: 'dpm' : is not a memb.....

--gaosi123

5. Re:Fast RCNN 训练自己数据集 (2修改数据读取接口)

@楼棘航的blog楼主你好！我在EdgeBoxes提取OP的时候也是直接用的默认参数，并且将坐标[x y w h]变成了左上右下的形式，但是发现检测车的时候效果并没有Selective Search好，.....

–JustJay

```
bool is_shared_; //标记该layer是否被其他nets所共享
shared_ptr<boost::mutex> forward_mutex_;//若该layer被shared，则需要这个mutex序列保持forward过程的正常运行
void InitMutex();//初始化forward 的 mutex
void Lock();//locak mutex
void Unlock();//unlock mutex这一看就明白了
```

最后其实proto里面有个layer_parameter是挺重要的一个结构，存储了layer的大量信息，这个具体可以到proto文件夹下查看，这里就暂时不列出了。

楼棘航的blog
关注 - 1
粉丝 - 60
+加关注

« 上一篇：Caffe源码解析2：SycedMem
» 下一篇：Caffe源码解析4：Data_layer

posted @ 2016-01-22 23:48 楼棘航的blog 阅读(418) 评论(0) 编辑 收藏