caffe中HingeLossLayer层原理以及源码分析

时间：2015-06-27 18:23:32 阅读：66 评论：0 收藏：0 [点我收藏+]

标签：caffe hingeloss layer 原理 源码

输入：

bottom[0]: NxKx1x1维，N为样本个数，K为类别数。是预测值。

bottom[1]: Nx1x1x1维， N为样本个数，类别为K时，每个元素的取值范围为[0,1,2,...,K-1]。是groundTruth。

输出：

top[0]: 1x1x1x1维， 求得是hingeLoss。

关于HingeLoss：

p: 范数，默认是L1范数，可以在配置中设置为L1或者L2范数。

$\delta\{l_n = k\}$: 指示函数，如果第n个样本的真实label为k，则为，否则为-1。

tnk: bottom[0]中第n个样本，第k维的预测值。

前向传播代码分析：

```
template
void HingeLossLayer::Forward_cpu(const vector*>& bottom,
    const vector*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();  //得到num个样本的dim个预测值
  Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
  const Dtype* label = bottom[1]->cpu_data();              //得到num个样本的groundTruth
  int num = bottom[0]->num();
  int count = bottom[0]->count();
  int dim = count / num;
  caffe_copy(count, bottom_data, bottom_diff);
  for (int i = 0; i < num; ++i) {
    //label[i]中存储了第i个样本的真实class，取值范围[0,1,2,...,K-1]
    //此处将第i个样本的K维预测值的label[i]处乘以-1相当于计算
   //caffe中HingeLossLayer层原理以及源码分析
    bottom_diff[i * dim + static_cast(label[i])] *= -1;
  }
  for (int i = 0; i < num; ++i) {
    for (int j = 0; j < dim; ++j) {
      //计算 caffe中HingeLossLayer层原理以及源码分析，存入 bottom_diff，即bottom[0]->mutable_cpu_
diff()中
      bottom_diff[i * dim + j] = std::max( Dtype(0), 1 + bottom_diff[i * dim + j]);
    }
```

```
  }
  Dtype* loss = top[0]->mutable_cpu_data();
  switch (this->layer_param_.hinge_loss_param().norm()) {
  case HingeLossParameter_Norm_L1:  //L1范数
    loss[0] = caffe_cpu_asum(count, bottom_diff) / num;
    break;
  case HingeLossParameter_Norm_L2: //L2范数
    loss[0] = caffe_cpu_dot(count, bottom_diff, bottom_diff) / num;
    break;
  default:
    LOG(FATAL) << "Unknown Norm";
  }
}
```

反向传播原理：

由于bottom[1]是groundtruth，不需要反传，只需要对bottom[0]进行反传，反传是损失E对t的偏导。

以L2范数为例，求偏导为：

$$记\max(0,1-\delta\{l_n=k\}t_{nk})为\text{hinge},\quad \frac{2}{\mathrm{N}}\bullet hinge\bullet\frac{\partial hinge}{\partial t_{nk}}$$

caffe中HingeLossLayer层原理以及源码分析

其中：

$$\frac{\partial hinge}{\partial t_{nk}}=\begin{cases}0, & hinge=0\\ -1, & hinge>0\end{cases}$$

caffe中HingeLossLayer层原理以及源码分析

反向传播源码分析：

```
template
void HingeLossLayer::Backward_cpu(const vector*>& top,
    const vector& propagate_down, const vector*>& bottom) {
  if (propagate_down[1]) {
    LOG(FATAL) << this->type()
               << " Layer cannot backpropagate to label inputs.";
  }
  if (propagate_down[0]) {
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff(); //说明中提到的hinge
    const Dtype* label = bottom[1]->cpu_data();
    int num = bottom[0]->num();
    int count = bottom[0]->count();
    int dim = count / num;
    for (int i = 0; i < num; ++i) {
      //相当于求hinge*偏hinge/偏tnk部分
      bottom_diff[i * dim + static_cast(label[i])] *= -1;
    }
    const Dtype loss_weight = top[0]->cpu_diff()[0];
    switch (this->layer_param_.hinge_loss_param().norm()) {
    case HingeLossParameter_Norm_L1:  //L1部分反传
      caffe_cpu_sign(count, bottom_diff, bottom_diff);  //L1求导的结果: 正返回1 负返回-1 0返回0
```

```
    caffe_scal(count, loss_weight / num, bottom_diff); //scale一下
    break;
  case HingeLossParameter_Norm_L2: //L2部分反传，就是scale一下
    caffe_scal(count, loss_weight * 2 / num, bottom_diff);
    break;
  default:
    LOG(FATAL) << "Unknown Norm";
  }
 }
}
```


caffe中HingeLossLayer层原理以及源码分析

标签：caffe hingeloss layer 原理 源码