如何在caffe中增加layer以及caffe中triple loss layer的实现

标签：caffe triple loss layer 实现

关于triplet loss的原理，目标函数和梯度推导在上一篇博客中已经讲过了，具体见：triplet loss原理以及梯度推导，这篇博文主要是讲caffe下实现triplet loss，编程菜鸟，如果有写的不优化的地方，欢迎指出。

1.如何在caffe中增加新的layer

新版的caffe中增加新的layer，变得轻松多了，概括说来，分四步：

1) 在./src/caffe/proto/caffe.proto 中增加 对应layer的paramter message；

2) 在./include/caffe/***layers.hpp中增加该layer的类的声明，***表示有common_layers.hpp, data_layers.hpp, neuron_layers.hpp, vision_layers.hpp 和loss_layers.hpp等；

3) 在./src/caffe/layers/目录下新建.cpp和.cu文件，进行类实现。

4) 在./src/caffe/gtest/中增加layer的测试代码，对所写的layer前传和反传进行测试，测试还包括速度。

最后一步很多人省了，或者没意识到，但是为保证代码正确，建议还是严格进行测试，磨刀不误砍柴功。

2.caffe中实现triplet loss layer

1.caffe.proto中增加triplet loss layer的定义

首先在message LayerParameter中追加 optional TripleLossParameter triple_loss_param = 138; 其中138是我目前LayerParameter message中现有元素的个数，具体是多少，可以看LayerParameter message上面注释中的：

//LayerParameter next available layer-specific ID: 134 (last added: reshape_param)

然后增加Message：

```
message TripletLossParameter {
    // margin for dissimilar pair
   optional float margin = 1 [default = 1.0];
}
```

其中 margin就是定义triplet loss原理以及梯度推导所讲的alpha。

2.在./include/caffe/loss_layers.hpp中增加triplet loss layer的类的声明

具体解释见注释，主要的是定义了一些变量，用来在前传中存储中间计算结果，以便在反传的时候避免重复计算。

```
/**
 * @brief Computes the triplet loss
 */
template <typename Dtype>
class TripletLossLayer : public LossLayer<Dtype> {
 public:
  explicit TripletLossLayer(const LayerParameter& param)
      : LossLayer<Dtype>(param){}
  virtual void LayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top);

  virtual inline int ExactNumBottomBlobs() const { return 4; }
  virtual inline const char* type() const { return "TripletLoss"; }
  /**
   * Unlike most loss layers, in the TripletLossLayer we can backpropagate
   * to the first three inputs.
   */
  virtual inline bool AllowForceBackward(const int bottom_index) const {
    return bottom_index != 3;
  }

 protected:
  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);
  virtual void Backward_gpu(const vector<Blob<Dtype>*>& top,
      const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom);

  Blob<Dtype> diff_ap_;  // cached for backward pass
  Blob<Dtype> diff_an_;  // cached for backward pass
  Blob<Dtype> diff_pn_;  // cached for backward pass

  Blob<Dtype> diff_sq_ap_;  // cached for backward pass
  Blob<Dtype> diff_sq_an_;  // tmp storage for gpu forward pass

  Blob<Dtype> dist_sq_ap_;  // cached for backward pass
  Blob<Dtype> dist_sq_an_;  // cached for backward pass

  Blob<Dtype> summer_vec_;  // tmp storage for gpu forward pass
  Blob<Dtype> dist_binary_;  // tmp storage for gpu forward pass
};
```

3. 在./src/caffe/layers/目录下新建triplet_loss_layer.cpp,实现类

主要实现三个功能：

LayerSetUp：主要是做一些CHECK工作，然后根据bottom和top对类中的数据成员初始化。

Forward_cpu：前传，计算loss

Backward_cpu：反传，计算梯度。

```
/*
 * triple_loss_layer.cpp
 *
 *  Created on: Jun 2, 2015
 *      Author: tangwei
 */

#include <algorithm>
#include <vector>

#include "caffe/layer.hpp"
#include "caffe/loss_layers.hpp"
#include "caffe/util/io.hpp"
#include "caffe/util/math_functions.hpp"

namespace caffe {

template <typename Dtype>
void TripletLossLayer<Dtype>::LayerSetUp(
  const vector<Blob<Dtype>*>& bottom, const vector<Blob<Dtype>*>& top) {
  LossLayer<Dtype>::LayerSetUp(bottom, top);
  CHECK_EQ(bottom[0]->num(), bottom[1]->num());
  CHECK_EQ(bottom[1]->num(), bottom[2]->num());
  CHECK_EQ(bottom[0]->channels(), bottom[1]->channels());
  CHECK_EQ(bottom[1]->channels(), bottom[2]->channels());
  CHECK_EQ(bottom[0]->height(), 1);
  CHECK_EQ(bottom[0]->width(), 1);
  CHECK_EQ(bottom[1]->height(), 1);
  CHECK_EQ(bottom[1]->width(), 1);
  CHECK_EQ(bottom[2]->height(), 1);
  CHECK_EQ(bottom[2]->width(), 1);

  CHECK_EQ(bottom[3]->channels(),1);
  CHECK_EQ(bottom[3]->height(), 1);
  CHECK_EQ(bottom[3]->width(), 1);

  diff_ap_.Reshape(bottom[0]->num(), bottom[0]->channels(), 1, 1);
  diff_an_.Reshape(bottom[0]->num(), bottom[0]->channels(), 1, 1);
  diff_pn_.Reshape(bottom[0]->num(), bottom[0]->channels(), 1, 1);

  diff_sq_ap_.Reshape(bottom[0]->num(), bottom[0]->channels(), 1, 1);
  diff_sq_an_.Reshape(bottom[0]->num(), bottom[0]->channels(), 1, 1);
  dist_sq_ap_.Reshape(bottom[0]->num(), 1, 1, 1);
  dist_sq_an_.Reshape(bottom[0]->num(), 1, 1, 1);
```

```
  // vector of ones used to sum along channels
  summer_vec_.Reshape(bottom[0]->channels(), 1, 1, 1);
  for (int i = 0; i < bottom[0]->channels(); ++i)
        summer_vec_.mutable_cpu_data()[i] = Dtype(1);
  dist_binary_.Reshape(bottom[0]->num(), 1, 1, 1);
   for (int i = 0; i < bottom[0]->num(); ++i)
      dist_binary_.mutable_cpu_data()[i] = Dtype(1);
}

template <typename Dtype>
void TripletLossLayer<Dtype>::Forward_cpu(
   const vector<Blob<Dtype>*>& bottom,
   const vector<Blob<Dtype>*>& top) {
  int count = bottom[0]->count();
  const Dtype* sampleW = bottom[3]->cpu_data();
  caffe_sub(
      count,
      bottom[0]->cpu_data(),  // a
      bottom[1]->cpu_data(),  // p
      diff_ap_.mutable_cpu_data());  // a_i-p_i
  caffe_sub(
       count,
       bottom[0]->cpu_data(),  // a
       bottom[2]->cpu_data(),  // n
       diff_an_.mutable_cpu_data());  // a_i-n_i
  caffe_sub(
       count,
       bottom[1]->cpu_data(),  // p
       bottom[2]->cpu_data(),  // n
       diff_pn_.mutable_cpu_data());  // p_i-n_i
  const int channels = bottom[0]->channels();
  Dtype margin = this->layer_param_.triple_loss_param().margin();

  Dtype loss(0.0);
  for (int i = 0; i < bottom[0]->num(); ++i) {
    dist_sq_ap_.mutable_cpu_data()[i] = caffe_cpu_dot(channels,
        diff_ap_.cpu_data() + (i*channels), diff_ap_.cpu_data() + (i*channels));
    dist_sq_an_.mutable_cpu_data()[i] = caffe_cpu_dot(channels,
        diff_an_.cpu_data() + (i*channels), diff_an_.cpu_data() + (i*channels));
    Dtype mdist = sampleW[i]*std::max(margin + dist_sq_ap_.cpu_data()[i] - dist_sq_an_.cpu_data()[i],
Dtype(0.0));
    loss += mdist;
    if(mdist==Dtype(0)){
        //dist_binary_.mutable_cpu_data()[i] = Dtype(0);
        //prepare for backward pass
        caffe_set(channels, Dtype(0), diff_ap_.mutable_cpu_data() + (i*channels));
        caffe_set(channels, Dtype(0), diff_an_.mutable_cpu_data() + (i*channels));
        caffe_set(channels, Dtype(0), diff_pn_.mutable_cpu_data() + (i*channels));
```

```
    }
  }
  loss = loss / static_cast<Dtype>(bottom[0]->num()) / Dtype(2);
  top[0]->mutable_cpu_data()[0] = loss;
}

template <typename Dtype>
void TripletLossLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
  //Dtype margin = this->layer_param_.contrastive_loss_param().margin();
  const Dtype* sampleW = bottom[3]->cpu_data();
  for (int i = 0; i < 3; ++i) {
    if (propagate_down[i]) {
      const Dtype sign = (i < 2) ? -1 : 1;
      const Dtype alpha = sign * top[0]->cpu_diff()[0] /
          static_cast<Dtype>(bottom[i]->num());
      int num = bottom[i]->num();
      int channels = bottom[i]->channels();
      for (int j = 0; j < num; ++j) {
        Dtype* bout = bottom[i]->mutable_cpu_diff();
        if (i==0) {  // a
          //if(dist_binary_.cpu_data()[j]>Dtype(0)){
                    caffe_cpu_axpby(
                        channels,
                        alpha*sampleW[j],
                        diff_pn_.cpu_data() + (j*channels),
                        Dtype(0.0),
                        bout + (j*channels));
          //}else{
          //  caffe_set(channels, Dtype(0), bout + (j*channels));
          //}
        } else if (i==1) {  // p
          //if(dist_binary_.cpu_data()[j]>Dtype(0)){
                    caffe_cpu_axpby(
                        channels,
                        alpha*sampleW[j],
                        diff_ap_.cpu_data() + (j*channels),
                        Dtype(0.0),
                        bout + (j*channels));
          //}else{
          //    caffe_set(channels, Dtype(0), bout + (j*channels));
          //}
              } else if (i==2) {  // n
                //if(dist_binary_.cpu_data()[j]>Dtype(0)){
                    caffe_cpu_axpby(
                        channels,
                        alpha*sampleW[j],
                        diff_an_.cpu_data() + (j*channels),
```

```
                        Dtype(0.0),
                        bout + (j*channels));
                //}else{
                //   caffe_set(channels, Dtype(0), bout + (j*channels));
                //}
              }
    } // for num
   } //if propagate_down[i]
 } //for i
}

#ifdef CPU_ONLY
STUB_GPU(TripleLossLayer);
#endif

INSTANTIATE_CLASS(TripletLossLayer);
REGISTER_LAYER_CLASS(TripletLoss);

}  // namespace caffe
```

4.在./src/caffe/layers/目录下新建triplet_loss_layer.cu,实现GPU下的前传和反传

在GPU下实现前传和反传

```
/*
 * triplet_loss_layer.cu
 *
 *  Created on: Jun 2, 2015
 *      Author: tangwei
 */

#include <algorithm>
#include <vector>

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

namespace caffe {

template <typename Dtype>
void TripletLossLayer<Dtype>::Forward_gpu(
    const vector<Blob<Dtype>*>& bottom, const vector<Blob<Dtype>*>& top) {
  const int count = bottom[0]->count();
  caffe_gpu_sub(
      count,
      bottom[0]->gpu_data(),  // a
```

```
    bottom[1]->gpu_data(),  // p
    diff_ap_.mutable_gpu_data());  // a_i-p_i
  caffe_gpu_sub(
        count,
        bottom[0]->gpu_data(),  // a
        bottom[2]->gpu_data(),  // n
        diff_an_.mutable_gpu_data());  // a_i-n_i
  caffe_gpu_sub(
    count,
    bottom[1]->gpu_data(),  // p
    bottom[2]->gpu_data(),  // n
    diff_pn_.mutable_gpu_data());  // p_i-n_i

  caffe_gpu_powx(
    count,
    diff_ap_.mutable_gpu_data(),  // a_i-p_i
    Dtype(2),
    diff_sq_ap_.mutable_gpu_data());  // (a_i-p_i)^2
  caffe_gpu_gemv(
    CblasNoTrans,
    bottom[0]->num(),
    bottom[0]->channels(),
    Dtype(1.0),                                 //alpha
    diff_sq_ap_.gpu_data(),  // (a_i-p_i)^2             // A
    summer_vec_.gpu_data(),                         // x
    Dtype(0.0),                                 //belta
    dist_sq_ap_.mutable_gpu_data());  // \Sum (a_i-p_i)^2  //y

  caffe_gpu_powx(
      count,
      diff_an_.mutable_gpu_data(),  // a_i-n_i
      Dtype(2),
      diff_sq_an_.mutable_gpu_data());  // (a_i-n_i)^2
  caffe_gpu_gemv(
      CblasNoTrans,
      bottom[0]->num(),
      bottom[0]->channels(),
      Dtype(1.0),                                 //alpha
      diff_sq_an_.gpu_data(),  // (a_i-n_i)^2             // A
      summer_vec_.gpu_data(),                         // x
      Dtype(0.0),                                 //belta
      dist_sq_an_.mutable_gpu_data());  // \Sum (a_i-n_i)^2  //y

  Dtype margin = this->layer_param_.triple_loss_param().margin();
  Dtype loss(0.0);
  const Dtype* sampleW = bottom[3]->cpu_data();
  for (int i = 0; i < bottom[0]->num(); ++i) {
    loss += sampleW[i]*std::max(margin +dist_sq_ap_.cpu_data()[i]- dist_sq_an_.cpu_data()[i], Dtype(0.
```

```
0));
  }
  loss = loss / static_cast<Dtype>(bottom[0]->num()) / Dtype(2);
  top[0]->mutable_cpu_data()[0] = loss;
}

template <typename Dtype>
__global__ void CLLBackward(const int count, const int channels,
    const Dtype margin, const Dtype alpha, const Dtype* sampleW,
    const Dtype* diff, const Dtype* dist_sq_ap_, const Dtype* dist_sq_an_,
    Dtype *bottom_diff) {
  CUDA_KERNEL_LOOP(i, count) {
    int n = i / channels;  // the num index, to access dist_sq_ap_ and dist_sq_an_
    Dtype mdist(0.0);
    mdist = margin +dist_sq_ap_[n] - dist_sq_an_[n];
    if (mdist > 0.0) {
            bottom_diff[i] = alpha*sampleW[n]*diff[i];
       } else {
            bottom_diff[i] = 0;
    }
  }
}

template <typename Dtype>
void TripletLossLayer<Dtype>::Backward_gpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
  Dtype margin = this->layer_param_.triple_loss_param().margin();
  const int count = bottom[0]->count();
  const int channels = bottom[0]->channels();

  for (int i = 0; i < 3; ++i) {
    if (propagate_down[i]) {
      const Dtype sign = (i < 2) ? -1 : 1;
      const Dtype alpha = sign * top[0]->cpu_diff()[0] /
          static_cast<Dtype>(bottom[0]->num());
      if(i==0){
            // NOLINT_NEXT_LINE(whitespace/operators)
            CLLBackward<Dtype><<<CAFFE_GET_BLOCKS(count), CAFFE_CUDA_NUM_THREADS>>>
(
                    count, channels, margin, alpha,
                    bottom[3]->gpu_data(),
                    diff_pn_.gpu_data(),  // the cached eltwise difference between p and n
                    dist_sq_ap_.gpu_data(),  // the cached square distance between a and p
                    dist_sq_an_.gpu_data(),  // the cached square distance between a and n
                    bottom[i]->mutable_gpu_diff());
            CUDA_POST_KERNEL_CHECK;
      }else if(i==1){
          // NOLINT_NEXT_LINE(whitespace/operators)
```

```
            CLLBackward<Dtype><<<CAFFE_GET_BLOCKS(count), CAFFE_CUDA_NUM_THREADS>>>
(
                    count, channels, margin, alpha,
                    bottom[3]->gpu_data(),
                    diff_ap_.gpu_data(),  // the cached eltwise difference between a and p
                    dist_sq_ap_.gpu_data(),  // the cached square distance between a and p
                    dist_sq_an_.gpu_data(),  // the cached square distance between a and n
                    bottom[i]->mutable_gpu_diff());
            CUDA_POST_KERNEL_CHECK;
    }else if(i==2){
      // NOLINT_NEXT_LINE(whitespace/operators)
            CLLBackward<Dtype><<<CAFFE_GET_BLOCKS(count), CAFFE_CUDA_NUM_THREADS>>>
(
                    count, channels, margin, alpha,
                    bottom[3]->gpu_data(),
                    diff_an_.gpu_data(),  // the cached eltwise difference between a and n
                    dist_sq_ap_.gpu_data(),  // the cached square distance between a and p
                    dist_sq_an_.gpu_data(),  // the cached square distance between a and n
                    bottom[i]->mutable_gpu_diff());
            CUDA_POST_KERNEL_CHECK;

    }
  }
 }
}

INSTANTIATE_LAYER_GPU_FUNCS(TripletLossLayer);

}  // namespace caffe
```

5. 在./src/caffe/test/目录下增加test_triplet_loss_layer.cpp

```
/*
 * test_triple_loss_layer.cpp
 *
 *  Created on: Jun 3, 2015
 *      Author: tangwei
 */

#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <cstring>
#include <vector>

#include "gtest/gtest.h"

#include "caffe/blob.hpp"
#include "caffe/common.hpp"
```

```
#include "caffe/filler.hpp"
#include "caffe/vision_layers.hpp"

#include "caffe/test/test_caffe_main.hpp"
#include "caffe/test/test_gradient_check_util.hpp"

namespace caffe {

template <typename TypeParam>
class TripletLossLayerTest : public MultiDeviceTest<TypeParam> {
 typedef typename TypeParam::Dtype Dtype;

 protected:
 TripletLossLayerTest()
     : blob_bottom_data_i_(new Blob<Dtype>(512, 2, 1, 1)),
       blob_bottom_data_j_(new Blob<Dtype>(512, 2, 1, 1)),
       blob_bottom_data_k_(new Blob<Dtype>(512, 2, 1, 1)),
       blob_bottom_y_(new Blob<Dtype>(512, 1, 1, 1)),
       blob_top_loss_(new Blob<Dtype>()) {
   // fill the values
   FillerParameter filler_param;
   filler_param.set_min(-1.0);
   filler_param.set_max(1.0);  // distances~=1.0 to test both sides of margin
   UniformFiller<Dtype> filler(filler_param);
   filler.Fill(this->blob_bottom_data_i_);
   blob_bottom_vec_.push_back(blob_bottom_data_i_);
   filler.Fill(this->blob_bottom_data_j_);
   blob_bottom_vec_.push_back(blob_bottom_data_j_);
   filler.Fill(this->blob_bottom_data_k_);
   blob_bottom_vec_.push_back(blob_bottom_data_k_);
   for (int i = 0; i < blob_bottom_y_->count(); ++i) {
     blob_bottom_y_->mutable_cpu_data()[i] = caffe_rng_rand() % 2;  // 0 or 1
   }
   blob_bottom_vec_.push_back(blob_bottom_y_);
   blob_top_vec_.push_back(blob_top_loss_);
 }
 virtual ~TripletLossLayerTest() {
   delete blob_bottom_data_i_;
   delete blob_bottom_data_j_;
   delete blob_bottom_data_k_;
   delete blob_top_loss_;
 }

  Blob<Dtype>* const blob_bottom_data_i_;
  Blob<Dtype>* const blob_bottom_data_j_;
  Blob<Dtype>* const blob_bottom_data_k_;
  Blob<Dtype>* const blob_bottom_y_;
  Blob<Dtype>* const blob_top_loss_;
```

```
  vector<Blob<Dtype>*> blob_bottom_vec_;
  vector<Blob<Dtype>*> blob_top_vec_;
};

TYPED_TEST_CASE(TripletLossLayerTest, TestDtypesAndDevices);

TYPED_TEST(TripletLossLayerTest, TestForward) {
  typedef typename TypeParam::Dtype Dtype;
  LayerParameter layer_param;
  TripletLossLayer<Dtype> layer(layer_param);
  layer.SetUp(this->blob_bottom_vec_, this->blob_top_vec_);
  layer.Forward(this->blob_bottom_vec_, this->blob_top_vec_);
  // manually compute to compare
  const Dtype margin = layer_param.triple_loss_param().margin();
  const int num = this->blob_bottom_data_i_->num();
  const int channels = this->blob_bottom_data_i_->channels();
  Dtype loss(0);
  for (int i = 0; i < num; ++i) {
    Dtype dist_sq_ij(0);
    Dtype dist_sq_ik(0);
    for (int j = 0; j < channels; ++j) {
      Dtype diff_ij = this->blob_bottom_data_i_->cpu_data()[i*channels+j] -
          this->blob_bottom_data_j_->cpu_data()[i*channels+j];
      dist_sq_ij += diff_ij*diff_ij;
      Dtype diff_ik = this->blob_bottom_data_i_->cpu_data()[i*channels+j] -
          this->blob_bottom_data_k_->cpu_data()[i*channels+j];
      dist_sq_ik += diff_ik*diff_ik;
    }
    loss += std::max(Dtype(0.0), margin+dist_sq_ij-dist_sq_ik);
  }
  loss /= static_cast<Dtype>(num) * Dtype(2);
  EXPECT_NEAR(this->blob_top_loss_->cpu_data()[0], loss, 1e-6);
}

TYPED_TEST(TripletLossLayerTest, TestGradient) {
  typedef typename TypeParam::Dtype Dtype;
  LayerParameter layer_param;
  TripletLossLayer<Dtype> layer(layer_param);
  layer.SetUp(this->blob_bottom_vec_, this->blob_top_vec_);
  GradientChecker<Dtype> checker(1e-2, 1e-2, 1701);
  // check the gradient for the first two bottom layers
  checker.CheckGradientExhaustive(&layer, this->blob_bottom_vec_,
      this->blob_top_vec_, 0);
  checker.CheckGradientExhaustive(&layer, this->blob_bottom_vec_,
      this->blob_top_vec_, 1);
}

}  // namespace caffe
```

3.编译测试

重新 make all 如果出错，检查代码语法错误。

make test

make runtest 如果成功，全是绿色的OK 否则会给出红色提示，就得看看是不是实现逻辑上出错了。

如何在caffe中增加layer以及caffe中triple loss layer的实现

标签：caffe triple loss layer 实现