[**Caffe-代码解析-compute\_image\_mean**](http://blog.csdn.net/chenriwei2/article/details/46362851)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-04 15:51 375人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46362851#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46362851#report)

功能：   
计算训练数据库的平均图像。   
因为平均归一化训练图像会对结果有提升，所以Caffe里面，提供了一个可选项。

使用方法：   
compute\_image\_mean [FLAGS] INPUT\_DB [OUTPUT\_FILE]\n”)   
参数：INPUT\_DB： 数据库   
参数（可选）：OUTPUT\_FILE： 输出文件名,不提供的话，不保存平均图像blob

实现方法：

数据源：求平均图像的方法是直接从数据库（LevelDB或者LMDB）里面直接读取出来的，而不是直接用图像数据库里面求出，意味着，必须先进行图像到数据库的转换后，才能求平均图像这一步。

接下来就是遍历KV数据库的每一个值while (cursor->valid()) 将每一个数据值转换为Datum，datum.ParseFromString(cursor->value());

接着将Datum阶码到sum\_blob 中，sum\_blob 是一个num=1，channels=图像.channel,height=图像.height ,width=图像.width 的blob

累加：

sum\_blob.set\_data(i, sum\_blob.data(i) + (uint8\_t)data[i]);

* 1

最后求平均：

sum\_blob.set\_data(i, sum\_blob.data(i) / count);

存在的问题：上述代码只是先累加在处于数目求和，显然，如果需要求平均的图像的数目**相当多**的话，就有可能溢出（浮点溢出），

最后，如果要求简单一点的话，也可以直接求每个通道的平均值。   
源代码：//2015.06.04版本

#include <stdint.h>

#include <algorithm>

#include <string>

#include <utility>

#include <vector>

#include "boost/scoped\_ptr.hpp"

#include "gflags/gflags.h"

#include "glog/logging.h"

#include "caffe/proto/caffe.pb.h"

#include "caffe/util/db.hpp"

#include "caffe/util/io.hpp"

using namespace caffe; // NOLINT(build/namespaces)

using std::max;

using std::pair;

using boost::scoped\_ptr;

DEFINE\_string(backend, "lmdb",

"The backend {leveldb, lmdb} containing the images");

int main(int argc, char\*\* argv) {

::google::InitGoogleLogging(argv[0]);

#ifndef GFLAGS\_GFLAGS\_H\_

namespace gflags = google;

#endif

gflags::SetUsageMessage("Compute the mean\_image of a set of images given by"

" a leveldb/lmdb\n"

"Usage:\n"

" compute\_image\_mean [FLAGS] INPUT\_DB [OUTPUT\_FILE]\n");

gflags::ParseCommandLineFlags(&argc, &argv, true);

if (argc < 2 || argc > 3) {

gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/compute\_image\_mean");

return 1;

}

scoped\_ptr<db::DB> db(db::GetDB(FLAGS\_backend));

db->Open(argv[1], db::READ);

scoped\_ptr<db::Cursor> cursor(db->NewCursor());

BlobProto sum\_blob;

int count = 0;

// load first datum

Datum datum;

datum.ParseFromString(cursor->value());

if (DecodeDatumNative(&datum)) {

LOG(INFO) << "Decoding Datum";

}

sum\_blob.set\_num(1);

sum\_blob.set\_channels(datum.channels());

sum\_blob.set\_height(datum.height());

sum\_blob.set\_width(datum.width());

const int data\_size = datum.channels() \* datum.height() \* datum.width();

int size\_in\_datum = std::max<int>(datum.data().size(),

datum.float\_data\_size());

for (int i = 0; i < size\_in\_datum; ++i) {

sum\_blob.add\_data(0.);

}

LOG(INFO) << "Starting Iteration";

while (cursor->valid()) {

Datum datum;

datum.ParseFromString(cursor->value());

DecodeDatumNative(&datum);

const std::string& data = datum.data();

size\_in\_datum = std::max<int>(datum.data().size(),

datum.float\_data\_size());

CHECK\_EQ(size\_in\_datum, data\_size) << "Incorrect data field size " <<

size\_in\_datum;

if (data.size() != 0) {

CHECK\_EQ(data.size(), size\_in\_datum);

for (int i = 0; i < size\_in\_datum; ++i) {

sum\_blob.set\_data(i, sum\_blob.data(i) + (uint8\_t)data[i]);

}

} else {

CHECK\_EQ(datum.float\_data\_size(), size\_in\_datum);

for (int i = 0; i < size\_in\_datum; ++i) {

sum\_blob.set\_data(i, sum\_blob.data(i) +

static\_cast<float>(datum.float\_data(i)));

}

}

++count;

if (count % 10000 == 0) {

LOG(INFO) << "Processed " << count << " files.";

}

cursor->Next();

}

if (count % 10000 != 0) {

LOG(INFO) << "Processed " << count << " files.";

}

for (int i = 0; i < sum\_blob.data\_size(); ++i) {

sum\_blob.set\_data(i, sum\_blob.data(i) / count);

}

// Write to disk

if (argc == 3) {

LOG(INFO) << "Write to " << argv[2];

WriteProtoToBinaryFile(sum\_blob, argv[2]);

}

const int channels = sum\_blob.channels();

const int dim = sum\_blob.height() \* sum\_blob.width();

std::vector<float> mean\_values(channels, 0.0);

LOG(INFO) << "Number of channels: " << channels;

for (int c = 0; c < channels; ++c) {

for (int i = 0; i < dim; ++i) {

mean\_values[c] += sum\_blob.data(dim \* c + i);

}

LOG(INFO) << "mean\_value channel [" << c << "]:" << mean\_values[c] / dim;

}

return 0;

}

# [Caffe 代码解析-convert\_imageset](http://blog.csdn.net/chenriwei2/article/details/46361333)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-04 15:55 396人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46361333#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46361333#report)

**功能**：   
将图像数据，转化为KV数据库（LevelDB或者LMDB）   
需要提供文件列表（包含对应的标签）   
**使用方法：**   
convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME   
其中   
参数：ROOTFOLDER 表示输入的文件夹   
参数：LISTFILE 表示输入文件列表，其每一行为：类似 subfolder1/file1.JPEG 7   
可选参数：[FLAGS] 可以指示是否使用shuffle，颜色空间，编码等。

**实现方法：**   
首先，将文件名与它对应的标签用 std::pair 存储起来，其中first存储文件名，second存储标签，

其次，数据通过 Datum datum来存储，将图像与标签转为Datum 需要通过函数ReadImageToDatum（） 来完成，

再次， Datum 数据又是通过datum.SerializeToString(&out)把数据序列化为字符串 string out;，

最后， 将字符串 string out ，通过txn->Put(string(key\_cstr, length), out)写入数据库DB。

源代码//2015.06.04版本

// This program converts a set of images to a lmdb/leveldb by storing them

// as Datum proto buffers.

// Usage:

// convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME

//

// where ROOTFOLDER is the root folder that holds all the images, and LISTFILE

// should be a list of files as well as their labels, in the format as

// subfolder1/file1.JPEG 7

// ....

#include <algorithm>

#include <fstream> // NOLINT(readability/streams)

#include <string>

#include <utility>

#include <vector>

#include "boost/scoped\_ptr.hpp"

#include "gflags/gflags.h"

#include "glog/logging.h"

#include "caffe/proto/caffe.pb.h"

#include "caffe/util/db.hpp"

#include "caffe/util/io.hpp"

#include "caffe/util/rng.hpp"

using namespace caffe; // NOLINT(build/namespaces)

using std::pair;

using boost::scoped\_ptr;

DEFINE\_bool(gray, false,

"When this option is on, treat images as grayscale ones");

DEFINE\_bool(shuffle, false,

"Randomly shuffle the order of images and their labels");

DEFINE\_string(backend, "lmdb",

"The backend {lmdb, leveldb} for storing the result");

DEFINE\_int32(resize\_width, 0, "Width images are resized to");

DEFINE\_int32(resize\_height, 0, "Height images are resized to");

DEFINE\_bool(check\_size, false,

"When this option is on, check that all the datum have the same size");

DEFINE\_bool(encoded, false,

"When this option is on, the encoded image will be save in datum");

DEFINE\_string(encode\_type, "",

"Optional: What type should we encode the image as ('png','jpg',...).");

int main(int argc, char\*\* argv) {

::google::InitGoogleLogging(argv[0]);

#ifndef GFLAGS\_GFLAGS\_H\_

namespace gflags = google;

#endif

gflags::SetUsageMessage("Convert a set of images to the leveldb/lmdb\n"

"format used as input for Caffe.\n"

"Usage:\n"

" convert\_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB\_NAME\n"

"The ImageNet dataset for the training demo is at\n"

" http://www.image-net.org/download-images\n");

gflags::ParseCommandLineFlags(&argc, &argv, true);

if (argc < 4) {

gflags::ShowUsageWithFlagsRestrict(argv[0], "tools/convert\_imageset");

return 1;

}

const bool is\_color = !FLAGS\_gray;

const bool check\_size = FLAGS\_check\_size;

const bool encoded = FLAGS\_encoded;

const string encode\_type = FLAGS\_encode\_type;

std::ifstream infile(argv[2]);

std::vector<std::pair<std::string, int> > lines;

std::string filename;

int label;

while (infile >> filename >> label) {

lines.push\_back(std::make\_pair(filename, label));

}

if (FLAGS\_shuffle) {

// randomly shuffle data

LOG(INFO) << "Shuffling data";

shuffle(lines.begin(), lines.end());

}

LOG(INFO) << "A total of " << lines.size() << " images.";

if (encode\_type.size() && !encoded)

LOG(INFO) << "encode\_type specified, assuming encoded=true.";

int resize\_height = std::max<int>(0, FLAGS\_resize\_height);

int resize\_width = std::max<int>(0, FLAGS\_resize\_width);

// Create new DB

scoped\_ptr<db::DB> db(db::GetDB(FLAGS\_backend));

db->Open(argv[3], db::NEW);

scoped\_ptr<db::Transaction> txn(db->NewTransaction());

// Storing to db

std::string root\_folder(argv[1]);

Datum datum;

int count = 0;

const int kMaxKeyLength = 256;

char key\_cstr[kMaxKeyLength];

int data\_size = 0;

bool data\_size\_initialized = false;

for (int line\_id = 0; line\_id < lines.size(); ++line\_id) {

bool status;

std::string enc = encode\_type;

if (encoded && !enc.size()) {

// Guess the encoding type from the file name

string fn = lines[line\_id].first;

size\_t p = fn.rfind('.');

if ( p == fn.npos )

LOG(WARNING) << "Failed to guess the encoding of '" << fn << "'";

enc = fn.substr(p);

std::transform(enc.begin(), enc.end(), enc.begin(), ::tolower);

}

status = ReadImageToDatum(root\_folder + lines[line\_id].first,

lines[line\_id].second, resize\_height, resize\_width, is\_color,

enc, &datum);

if (status == false) continue;

if (check\_size) {

if (!data\_size\_initialized) {

data\_size = datum.channels() \* datum.height() \* datum.width();

data\_size\_initialized = true;

} else {

const std::string& data = datum.data();

CHECK\_EQ(data.size(), data\_size) << "Incorrect data field size "

<< data.size();

}

}

// sequential

int length = snprintf(key\_cstr, kMaxKeyLength, "%08d\_%s", line\_id,

lines[line\_id].first.c\_str());

// Put in db

string out;

CHECK(datum.SerializeToString(&out));

txn->Put(string(key\_cstr, length), out);

if (++count % 1000 == 0) {

// Commit db

txn->Commit();

txn.reset(db->NewTransaction());

LOG(ERROR) << "Processed " << count << " files.";

}

}

// write the last batch

if (count % 1000 != 0) {

txn->Commit();

LOG(ERROR) << "Processed " << count << " files.";

}

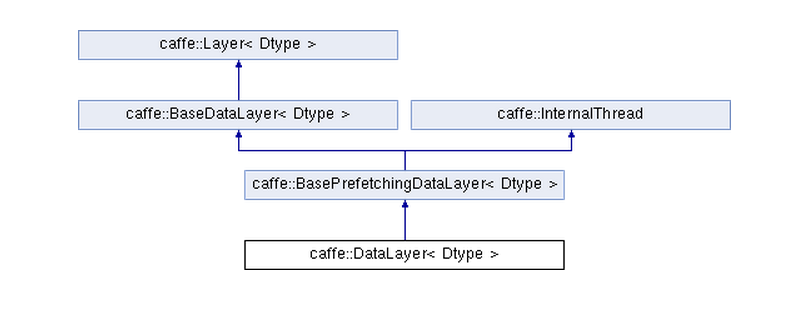
return 0;

}

[**Caffe-源码解析--DataLayer**](http://blog.csdn.net/chenriwei2/article/details/46366731)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-04 20:27 304人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46366731#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46366731#report)

函数功能：   
DataLayer 用于将数据库上的内容，一个batch一个batch的读入到相对应的Blob中，

首先查看其继承关系   


注意其不是直接继承于BaseDatalayer，因为，它需要并行的读取数据库上的数据，需要新开线程来预先读入数据。

DataLayer 有两个指针成员用来存放数据库和游标，

shared\_ptr<db::DB> db\_;

shared\_ptr<db::Cursor> cursor\_;

* 1
* 2

其继承自基类的成员变量有

protected:

Blob<Dtype> prefetch\_data\_;

Blob<Dtype> prefetch\_label\_;

Blob<Dtype> transformed\_data\_;

* 1
* 2
* 3
* 4

用于保存预读取的数据，标签，以及转换过的数据

继承自BaseDataLayer的成员变量有：   
bool output\_labels\_;

其成员函数InternalThreadEntry()用于真正的数据读入操作，

其中

Dtype\* top\_data = this->prefetch\_data\_.mutable\_cpu\_data();

Dtype\* top\_label = NULL; // suppress warnings about uninitialized variables

* 1
* 2

指针用于保留输入的批数据。   
数据库里面的数据依然是先转化为Datum，   
Datum datum;   
datum.ParseFromString(cursor\_->value());

int offset = this->prefetch\_data\_.offset(item\_id);   
this->transformed\_data\_.set\_cpu\_data(top\_data + offset);   
top\_label[item\_id] = datum.label();

其读取数据库输入也是通过游标来操作，cursor\_->Next();,注意这里都是按照顺序读入的，所以，需要自己保证在输入存入数据库的时候确保其是无序的。

=============   
源代码：

#include <opencv2/core/core.hpp>

#include <stdint.h>

#include <string>

#include <vector>

#include "caffe/common.hpp"

#include "caffe/data\_layers.hpp"

#include "caffe/layer.hpp"

#include "caffe/proto/caffe.pb.h"

#include "caffe/util/benchmark.hpp"

#include "caffe/util/io.hpp"

#include "caffe/util/math\_functions.hpp"

#include "caffe/util/rng.hpp"

namespace caffe {

template <typename Dtype>

DataLayer<Dtype>::~DataLayer<Dtype>() {

this->JoinPrefetchThread();

}

template <typename Dtype>

void DataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {

// Initialize DB

db\_.reset(db::GetDB(this->layer\_param\_.data\_param().backend()));

db\_->Open(this->layer\_param\_.data\_param().source(), db::READ);

cursor\_.reset(db\_->NewCursor());

// Check if we should randomly skip a few data points

if (this->layer\_param\_.data\_param().rand\_skip()) {

unsigned int skip = caffe\_rng\_rand() %

this->layer\_param\_.data\_param().rand\_skip();

LOG(INFO) << "Skipping first " << skip << " data points.";

while (skip-- > 0) {

cursor\_->Next();

}

}

// Read a data point, and use it to initialize the top blob.

Datum datum;

datum.ParseFromString(cursor\_->value());

bool force\_color = this->layer\_param\_.data\_param().force\_encoded\_color();

if ((force\_color && DecodeDatum(&datum, true)) ||

DecodeDatumNative(&datum)) {

LOG(INFO) << "Decoding Datum";

}

// image

int crop\_size = this->layer\_param\_.transform\_param().crop\_size();

if (crop\_size > 0) {

top[0]->Reshape(this->layer\_param\_.data\_param().batch\_size(),

datum.channels(), crop\_size, crop\_size);

this->prefetch\_data\_.Reshape(this->layer\_param\_.data\_param().batch\_size(),

datum.channels(), crop\_size, crop\_size);

this->transformed\_data\_.Reshape(1, datum.channels(), crop\_size, crop\_size);

} else {

top[0]->Reshape(

this->layer\_param\_.data\_param().batch\_size(), datum.channels(),

datum.height(), datum.width());

this->prefetch\_data\_.Reshape(this->layer\_param\_.data\_param().batch\_size(),

datum.channels(), datum.height(), datum.width());

this->transformed\_data\_.Reshape(1, datum.channels(),

datum.height(), datum.width());

}

LOG(INFO) << "output data size: " << top[0]->num() << ","

<< top[0]->channels() << "," << top[0]->height() << ","

<< top[0]->width();

// label

if (this->output\_labels\_) {

vector<int> label\_shape(1, this->layer\_param\_.data\_param().batch\_size());

top[1]->Reshape(label\_shape);

this->prefetch\_label\_.Reshape(label\_shape);

}

}

// This function is used to create a thread that prefetches the data.

template <typename Dtype>

void DataLayer<Dtype>::InternalThreadEntry() {

CPUTimer batch\_timer;

batch\_timer.Start();

double read\_time = 0;

double trans\_time = 0;

CPUTimer timer;

CHECK(this->prefetch\_data\_.count());

CHECK(this->transformed\_data\_.count());

// Reshape on single input batches for inputs of varying dimension.

const int batch\_size = this->layer\_param\_.data\_param().batch\_size();

const int crop\_size = this->layer\_param\_.transform\_param().crop\_size();

bool force\_color = this->layer\_param\_.data\_param().force\_encoded\_color();

if (batch\_size == 1 && crop\_size == 0) {

Datum datum;

datum.ParseFromString(cursor\_->value());

if (datum.encoded()) {

if (force\_color) {

DecodeDatum(&datum, true);

} else {

DecodeDatumNative(&datum);

}

}

this->prefetch\_data\_.Reshape(1, datum.channels(),

datum.height(), datum.width());

this->transformed\_data\_.Reshape(1, datum.channels(),

datum.height(), datum.width());

}

Dtype\* top\_data = this->prefetch\_data\_.mutable\_cpu\_data();

Dtype\* top\_label = NULL; // suppress warnings about uninitialized variables

if (this->output\_labels\_) {

top\_label = this->prefetch\_label\_.mutable\_cpu\_data();

}

for (int item\_id = 0; item\_id < batch\_size; ++item\_id) {

timer.Start();

// get a blob

Datum datum;

datum.ParseFromString(cursor\_->value());

cv::Mat cv\_img;

if (datum.encoded()) {

if (force\_color) {

cv\_img = DecodeDatumToCVMat(datum, true);

} else {

cv\_img = DecodeDatumToCVMatNative(datum);

}

if (cv\_img.channels() != this->transformed\_data\_.channels()) {

LOG(WARNING) << "Your dataset contains encoded images with mixed "

<< "channel sizes. Consider adding a 'force\_color' flag to the "

<< "model definition, or rebuild your dataset using "

<< "convert\_imageset.";

}

}

read\_time += timer.MicroSeconds();

timer.Start();

// Apply data transformations (mirror, scale, crop...)

int offset = this->prefetch\_data\_.offset(item\_id);

this->transformed\_data\_.set\_cpu\_data(top\_data + offset);

if (datum.encoded()) {

this->data\_transformer\_->Transform(cv\_img, &(this->transformed\_data\_));

} else {

this->data\_transformer\_->Transform(datum, &(this->transformed\_data\_));

}

if (this->output\_labels\_) {

top\_label[item\_id] = datum.label();

}

trans\_time += timer.MicroSeconds();

// go to the next iter

cursor\_->Next();

if (!cursor\_->valid()) {

DLOG(INFO) << "Restarting data prefetching from start.";

cursor\_->SeekToFirst();

}

}

batch\_timer.Stop();

DLOG(INFO) << "Prefetch batch: " << batch\_timer.MilliSeconds() << " ms.";

DLOG(INFO) << " Read time: " << read\_time / 1000 << " ms.";

DLOG(INFO) << "Transform time: " << trans\_time / 1000 << " ms.";

}

INSTANTIATE\_CLASS(DataLayer);

REGISTER\_LAYER\_CLASS(Data);

} // namespace caffe

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 153
* 154
* 155
* 156
* 157
* 158
* 159
* 160
* 161
* 162
* 163
* 164
* 165
* 166
* 167

其相对应的头文件信息：

template <typename Dtype>

class BaseDataLayer : public Layer<Dtype> {

public:

explicit BaseDataLayer(const LayerParameter& param);

virtual ~BaseDataLayer() {}

// LayerSetUp: implements common data layer setup functionality, and calls

// DataLayerSetUp to do special data layer setup for individual layer types.

// This method may not be overridden except by the BasePrefetchingDataLayer.

virtual void LayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top);

virtual void DataLayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {}

// Data layers have no bottoms, so reshaping is trivial.

virtual void Reshape(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) {}

protected:

TransformationParameter transform\_param\_;

shared\_ptr<DataTransformer<Dtype> > data\_transformer\_;

bool output\_labels\_;

};

template <typename Dtype>

class BasePrefetchingDataLayer :

public BaseDataLayer<Dtype>, public InternalThread {

public:

explicit BasePrefetchingDataLayer(const LayerParameter& param)

: BaseDataLayer<Dtype>(param) {}

virtual ~BasePrefetchingDataLayer() {}

// LayerSetUp: implements common data layer setup functionality, and calls

// DataLayerSetUp to do special data layer setup for individual layer types.

// This method may not be overridden.

void LayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top);

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 CreatePrefetchThread();

virtual void JoinPrefetchThread();

// The thread's function

virtual void InternalThreadEntry() {}

protected:

Blob<Dtype> prefetch\_data\_;

Blob<Dtype> prefetch\_label\_;

Blob<Dtype> transformed\_data\_;

};

template <typename Dtype>

[**Caffe-代码解析-Blob**](http://blog.csdn.net/chenriwei2/article/details/46367023)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-04 21:19 406人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46367023#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46367023#report)

**主要功能：**

Blob 是Caffe作为数据传输的媒介，无论是网络权重参数，还是输入数据，都是转化为Blob数据结构来存储，网络，求解器等都是直接与此结构打交道的。

其直观的可以把它看成一个有4纬的结构体（包含数据和梯度），而实际上，它们只是一维的指针而已，其4维结构通过shape属性得以计算出来（根据C语言的数据顺序）。

**其成员变量有：**

protected:

shared\_ptr<SyncedMemory> data\_;// 存放数据

shared\_ptr<SyncedMemory> diff\_;//存放梯度

vector<int> shape\_; //存放形状

int count\_; //数据个数

int capacity\_; //数据容量

* 1
* 2
* 3
* 4
* 5
* 6

**成员函数，见的最多的有：**

const Dtype\* cpu\_data() const; //cpu使用的数据

void set\_cpu\_data(Dtype\* data);//用数据块的值来blob里面的data。

const Dtype\* gpu\_data() const;//返回不可更改的指针，下同

const Dtype\* cpu\_diff() const;

const Dtype\* gpu\_diff() const;

Dtype\* mutable\_cpu\_data();//返回可更改的指针，下同

Dtype\* mutable\_gpu\_data();

Dtype\* mutable\_cpu\_diff();

Dtype\* mutable\_gpu\_diff();

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

总之，带mutable\_开头的意味着可以对返回的指针内容进行更改，而不带mutable\_开头的返回const 指针，不能对其指针的内容进行修改，

int offset(const int n, const int c = 0, const int h = 0,const int w = 0) const

// 通过n,c,h,w 4个参数来计算一维向量的偏移量。

Dtype data\_at(const int n, const int c, const int h,const int w) const//通过n,c,h,w 4个参数来来获取该向量位置上的值。

Dtype diff\_at(const int n, const int c, const int h,const int w) const//同上

inline const shared\_ptr<SyncedMemory>& data() const {

CHECK(data\_);

return data\_;//返回数据，不能修改

}

inline const shared\_ptr<SyncedMemory>& diff() const {

CHECK(diff\_);

return diff\_;//返回梯度，不能修改

}

Reshape(...)//reshape 有多种多态的实现，可以是四个数字，长度为四的vector，其它blob等。

if (count\_ > capacity\_) {

capacity\_ = count\_;

data\_.reset(new SyncedMemory(capacity\_ \* sizeof(Dtype)));

diff\_.reset(new SyncedMemory(capacity\_ \* sizeof(Dtype)));

}//当空间不够的时候，需要扩大容量，reset。

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26

源代码：

#ifndef CAFFE\_BLOB\_HPP\_

#define CAFFE\_BLOB\_HPP\_

#include <algorithm>

#include <string>

#include <vector>

#include "caffe/common.hpp"

#include "caffe/proto/caffe.pb.h"

#include "caffe/syncedmem.hpp"

#include "caffe/util/math\_functions.hpp"

const int kMaxBlobAxes = INT\_MAX;

namespace caffe {

/\*\*

\* @brief A wrapper around SyncedMemory holders serving as the basic

\* computational unit through which Layer%s, Net%s, and Solver%s

\* interact.

\*

\* TODO(dox): more thorough description.

\*/

template <typename Dtype>

class Blob {

public:

Blob()

: data\_(), diff\_(), count\_(0), capacity\_(0) {}

/// @brief Deprecated; use <code>Blob(const vector<int>& shape)</code>.

explicit Blob(const int num, const int channels, const int height,

const int width);

explicit Blob(const vector<int>& shape);

/// @brief Deprecated; use <code>Reshape(const vector<int>& shape)</code>.

void Reshape(const int num, const int channels, const int height,

const int width);

/\*\*

\* @brief Change the dimensions of the blob, allocating new memory if

\* necessary.

\*

\* This function can be called both to create an initial allocation

\* of memory, and to adjust the dimensions of a top blob during Layer::Reshape

\* or Layer::Forward. When changing the size of blob, memory will only be

\* reallocated if sufficient memory does not already exist, and excess memory

\* will never be freed.

\*

\* Note that reshaping an input blob and immediately calling Net::Backward is

\* an error; either Net::Forward or Net::Reshape need to be called to

\* propagate the new input shape to higher layers.

\*/

void Reshape(const vector<int>& shape);

void Reshape(const BlobShape& shape);

void ReshapeLike(const Blob& other);

inline string shape\_string() const {

ostringstream stream;

for (int i = 0; i < shape\_.size(); ++i) {

stream << shape\_[i] << " ";

}

stream << "(" << count\_ << ")";

return stream.str();

}

inline const vector<int>& shape() const { return shape\_; }

/\*\*

\* @brief Returns the dimension of the index-th axis (or the negative index-th

\* axis from the end, if index is negative).

\*

\* @param index the axis index, which may be negative as it will be

\* "canonicalized" using CanonicalAxisIndex.

\* Dies on out of range index.

\*/

inline int shape(int index) const {

return shape\_[CanonicalAxisIndex(index)];

}

inline int num\_axes() const { return shape\_.size(); }

inline int count() const { return count\_; }

/\*\*

\* @brief Compute the volume of a slice; i.e., the product of dimensions

\* among a range of axes.

\*

\* @param start\_axis The first axis to include in the slice.

\*

\* @param end\_axis The first axis to exclude from the slice.

\*/

inline int count(int start\_axis, int end\_axis) const {

CHECK\_LE(start\_axis, end\_axis);

CHECK\_GE(start\_axis, 0);

CHECK\_GE(end\_axis, 0);

CHECK\_LE(start\_axis, num\_axes());

CHECK\_LE(end\_axis, num\_axes());

int count = 1;

for (int i = start\_axis; i < end\_axis; ++i) {

count \*= shape(i);

}

return count;

}

/\*\*

\* @brief Compute the volume of a slice spanning from a particular first

\* axis to the final axis.

\*

\* @param start\_axis The first axis to include in the slice.

\*/

inline int count(int start\_axis) const {

return count(start\_axis, num\_axes());

}

/\*\*

\* @brief Returns the 'canonical' version of a (usually) user-specified axis,

\* allowing for negative indexing (e.g., -1 for the last axis).

\*

\* @param index the axis index.

\* If 0 <= index < num\_axes(), return index.

\* If -num\_axes <= index <= -1, return (num\_axes() - (-index)),

\* e.g., the last axis index (num\_axes() - 1) if index == -1,

\* the second to last if index == -2, etc.

\* Dies on out of range index.

\*/

inline int CanonicalAxisIndex(int axis\_index) const {

CHECK\_GE(axis\_index, -num\_axes())

<< "axis " << axis\_index << " out of range for " << num\_axes()

<< "-D Blob with shape " << shape\_string();

CHECK\_LT(axis\_index, num\_axes())

<< "axis " << axis\_index << " out of range for " << num\_axes()

<< "-D Blob with shape " << shape\_string();

if (axis\_index < 0) {

return axis\_index + num\_axes();

}

return axis\_index;

}

/// @brief Deprecated legacy shape accessor num: use shape(0) instead.

inline int num() const { return LegacyShape(0); }

/// @brief Deprecated legacy shape accessor channels: use shape(1) instead.

inline int channels() const { return LegacyShape(1); }

/// @brief Deprecated legacy shape accessor height: use shape(2) instead.

inline int height() const { return LegacyShape(2); }

/// @brief Deprecated legacy shape accessor width: use shape(3) instead.

inline int width() const { return LegacyShape(3); }

inline int LegacyShape(int index) const {

CHECK\_LE(num\_axes(), 4)

<< "Cannot use legacy accessors on Blobs with > 4 axes.";

CHECK\_LT(index, 4);

CHECK\_GE(index, -4);

if (index >= num\_axes() || index < -num\_axes()) {

// Axis is out of range, but still in [0, 3] (or [-4, -1] for reverse

// indexing) -- this special case simulates the one-padding used to fill

// extraneous axes of legacy blobs.

return 1;

}

return shape(index);

}

inline int offset(const int n, const int c = 0, const int h = 0,

const int w = 0) const {

CHECK\_GE(n, 0);

CHECK\_LE(n, num());

CHECK\_GE(channels(), 0);

CHECK\_LE(c, channels());

CHECK\_GE(height(), 0);

CHECK\_LE(h, height());

CHECK\_GE(width(), 0);

CHECK\_LE(w, width());

return ((n \* channels() + c) \* height() + h) \* width() + w;

}

inline int offset(const vector<int>& indices) const {

CHECK\_LE(indices.size(), num\_axes());

int offset = 0;

for (int i = 0; i < num\_axes(); ++i) {

offset \*= shape(i);

if (indices.size() > i) {

CHECK\_GE(indices[i], 0);

CHECK\_LT(indices[i], shape(i));

offset += indices[i];

}

}

return offset;

}

/\*\*

\* @brief Copy from a source Blob.

\*

\* @param source the Blob to copy from

\* @param copy\_diff if false, copy the data; if true, copy the diff

\* @param reshape if false, require this Blob to be pre-shaped to the shape

\* of other (and die otherwise); if true, Reshape this Blob to other's

\* shape if necessary

\*/

void CopyFrom(const Blob<Dtype>& source, bool copy\_diff = false,

bool reshape = false);

inline Dtype data\_at(const int n, const int c, const int h,

const int w) const {

return cpu\_data()[offset(n, c, h, w)];

}

inline Dtype diff\_at(const int n, const int c, const int h,

const int w) const {

return cpu\_diff()[offset(n, c, h, w)];

}

inline Dtype data\_at(const vector<int>& index) const {

return cpu\_data()[offset(index)];

}

inline Dtype diff\_at(const vector<int>& index) const {

return cpu\_diff()[offset(index)];

}

inline const shared\_ptr<SyncedMemory>& data() const {

CHECK(data\_);

return data\_;

}

inline const shared\_ptr<SyncedMemory>& diff() const {

CHECK(diff\_);

return diff\_;

}

const Dtype\* cpu\_data() const;

void set\_cpu\_data(Dtype\* data);

const Dtype\* gpu\_data() const;

const Dtype\* cpu\_diff() const;

const Dtype\* gpu\_diff() const;

Dtype\* mutable\_cpu\_data();

Dtype\* mutable\_gpu\_data();

Dtype\* mutable\_cpu\_diff();

Dtype\* mutable\_gpu\_diff();

void Update();

void FromProto(const BlobProto& proto, bool reshape = true);

void ToProto(BlobProto\* proto, bool write\_diff = false) const;

/// @brief Compute the sum of absolute values (L1 norm) of the data.

Dtype asum\_data() const;

/// @brief Compute the sum of absolute values (L1 norm) of the diff.

Dtype asum\_diff() const;

/// @brief Compute the sum of squares (L2 norm squared) of the data.

Dtype sumsq\_data() const;

/// @brief Compute the sum of squares (L2 norm squared) of the diff.

Dtype sumsq\_diff() const;

/// @brief Scale the blob data by a constant factor.

void scale\_data(Dtype scale\_factor);

/// @brief Scale the blob diff by a constant factor.

void scale\_diff(Dtype scale\_factor);

/\*\*

\* @brief Set the data\_ shared\_ptr to point to the SyncedMemory holding the

\* data\_ of Blob other -- useful in Layer%s which simply perform a copy

\* in their Forward pass.

\*

\* This deallocates the SyncedMemory holding this Blob's data\_, as

\* shared\_ptr calls its destructor when reset with the "=" operator.

\*/

void ShareData(const Blob& other);

/\*\*

\* @brief Set the diff\_ shared\_ptr to point to the SyncedMemory holding the

\* diff\_ of Blob other -- useful in Layer%s which simply perform a copy

\* in their Forward pass.

\*

\* This deallocates the SyncedMemory holding this Blob's diff\_, as

\* shared\_ptr calls its destructor when reset with the "=" operator.

\*/

void ShareDiff(const Blob& other);

bool ShapeEquals(const BlobProto& other);

protected:

shared\_ptr<SyncedMemory> data\_;

shared\_ptr<SyncedMemory> diff\_;

vector<int> shape\_;

int count\_;

int capacity\_;

DISABLE\_COPY\_AND\_ASSIGN(Blob);

}; // class Blob

} // namespace caffe

#endif // CAFFE\_BLOB\_HPP\_

[**Caffe-代码解析-SyncedMemory**](http://blog.csdn.net/chenriwei2/article/details/46368707)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-04 22:54 228人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46368707#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46368707#report)

**功能：**

Caffe的底层数据的切换（cpu模式和gpu模式），需要用到内存同步模块。

源码：头文件

#ifndef CAFFE\_SYNCEDMEM\_HPP\_

#define CAFFE\_SYNCEDMEM\_HPP\_

#include <cstdlib>

#include "caffe/common.hpp"

#include "caffe/util/math\_functions.hpp"

namespace caffe {

inline void CaffeMallocHost(void\*\* ptr, size\_t size) {

\*ptr = malloc(size);

CHECK(\*ptr) << "host allocation of size " << size << " failed";

}

inline void CaffeFreeHost(void\* ptr) {

free(ptr);

}

/\*\*

\* @brief Manages memory allocation and synchronization between the host (CPU)

\* and device (GPU).

\*

\* TODO(dox): more thorough description.

\*/

class SyncedMemory {

public:

SyncedMemory()

: cpu\_ptr\_(NULL), gpu\_ptr\_(NULL), size\_(0), head\_(UNINITIALIZED),

own\_cpu\_data\_(false) {}

explicit SyncedMemory(size\_t size)

: cpu\_ptr\_(NULL), gpu\_ptr\_(NULL), size\_(size), head\_(UNINITIALIZED),

own\_cpu\_data\_(false) {}

~SyncedMemory();

const void\* cpu\_data();//获取cpu数据，返回void \* 指针

void set\_cpu\_data(void\* data);//用一个void \* 指针修改指针

const void\* gpu\_data();//获取gpu数据，返回void \* 指针

void\* mutable\_cpu\_data();//获取可以更改cpu数据，返回void \* 指针

void\* mutable\_gpu\_data();//获取可以更改gpu数据，返回void \* 指针

enum SyncedHead { UNINITIALIZED, HEAD\_AT\_CPU, HEAD\_AT\_GPU, SYNCED };//enum枚举值

SyncedHead head() { return head\_; }//获得枚举值

size\_t size() { return size\_; }//获得数据大小

private:

void to\_cpu();//转为cpu模式

void to\_gpu(); //转为gpu模式

void\* cpu\_ptr\_;//指向cpu的指针

void\* gpu\_ptr\_;//指向gpu的指指针

size\_t size\_; //大小

SyncedHead head\_; //数据存放的位置，枚举值之一

bool own\_cpu\_data\_;//是否有cpu数据

DISABLE\_COPY\_AND\_ASSIGN(SyncedMemory);

}; // class SyncedMemory

} // namespace caffe

#endif // CAFFE\_SYNCEDMEM\_HPP\_

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58

实现文件：

#include <cstring>

#include "caffe/common.hpp"

#include "caffe/syncedmem.hpp"

#include "caffe/util/math\_functions.hpp"

namespace caffe {

//析构函数，调用caffe函数来释放空间

SyncedMemory::~SyncedMemory() {

if (cpu\_ptr\_ && own\_cpu\_data\_) {

CaffeFreeHost(cpu\_ptr\_);

}

//如果有gpu数据，也进行释放。

#ifndef CPU\_ONLY

if (gpu\_ptr\_) {

CUDA\_CHECK(cudaFree(gpu\_ptr\_));

}

#endif // CPU\_ONLY

}

// 根据head信息，选择：1， 分类cpu空间，2.拷贝gpu数据值cpu

inline void SyncedMemory::to\_cpu() {

switch (head\_) {

case UNINITIALIZED:

CaffeMallocHost(&cpu\_ptr\_, size\_);

caffe\_memset(size\_, 0, cpu\_ptr\_);

head\_ = HEAD\_AT\_CPU;

own\_cpu\_data\_ = true;

break;

case HEAD\_AT\_GPU:

#ifndef CPU\_ONLY

if (cpu\_ptr\_ == NULL) {

CaffeMallocHost(&cpu\_ptr\_, size\_);

own\_cpu\_data\_ = true;

}

caffe\_gpu\_memcpy(size\_, gpu\_ptr\_, cpu\_ptr\_);

head\_ = SYNCED;

#else

NO\_GPU;

#endif

break;

case HEAD\_AT\_CPU:

case SYNCED:

break;

}

}

// 根据head信息，选择：1， 分类gpu空间，2.拷贝cpu数据值gpu

inline void SyncedMemory::to\_gpu() {

#ifndef CPU\_ONLY

switch (head\_) {

case UNINITIALIZED:

CUDA\_CHECK(cudaMalloc(&gpu\_ptr\_, size\_));

caffe\_gpu\_memset(size\_, 0, gpu\_ptr\_);

head\_ = HEAD\_AT\_GPU;

break;

case HEAD\_AT\_CPU:

if (gpu\_ptr\_ == NULL) {

CUDA\_CHECK(cudaMalloc(&gpu\_ptr\_, size\_));

}

caffe\_gpu\_memcpy(size\_, cpu\_ptr\_, gpu\_ptr\_);

head\_ = SYNCED;

break;

case HEAD\_AT\_GPU:

case SYNCED:

break;

}

#else

NO\_GPU;

#endif

}

//返回cpu指针 void \* 类型

const void\* SyncedMemory::cpu\_data() {

to\_cpu();

return (const void\*)cpu\_ptr\_;

}

// 设置cpu数据，利用另外一个指针的数据来初始化

void SyncedMemory::set\_cpu\_data(void\* data) {

CHECK(data);

if (own\_cpu\_data\_) {

CaffeFreeHost(cpu\_ptr\_);

}

cpu\_ptr\_ = data;//直接重置指针，

head\_ = HEAD\_AT\_CPU;

own\_cpu\_data\_ = false;//设false

}

//获得gpu指针

const void\* SyncedMemory::gpu\_data() {

#ifndef CPU\_ONLY

to\_gpu();

return (const void\*)gpu\_ptr\_;

#else

NO\_GPU;

#endif

}

//获得可更改的cpu指针

void\* SyncedMemory::mutable\_cpu\_data() {

to\_cpu();

head\_ = HEAD\_AT\_CPU;

return cpu\_ptr\_;

}

//获得可更改的gpu指针

void\* SyncedMemory::mutable\_gpu\_data() {

#ifndef CPU\_ONLY

to\_gpu();

head\_ = HEAD\_AT\_GPU;

return gpu\_ptr\_;

#else

NO\_GPU;

#endif

}

} // namespace caffe

# [Caffe-代码解析-Layer](http://blog.csdn.net/chenriwei2/article/details/46417293)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-08 20:32 414人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46417293#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46417293#report)

**Layer 功能：**

是所有的网络层的基类，其中，定义了一些通用的接口，比如前馈，反馈，reshape，setup等。

#ifndef CAFFE\_LAYER\_H\_

#define CAFFE\_LAYER\_H\_

#include <algorithm>

#include <string>

#include <vector>

#include "caffe/blob.hpp"

#include "caffe/common.hpp"

#include "caffe/layer\_factory.hpp"

#include "caffe/proto/caffe.pb.h"

#include "caffe/util/device\_alternate.hpp"

namespace caffe {

// 功能：所有的网络层的基类，定义的所有的网络层的通用接口。

// 前馈接口，必须实现

// 反馈接口，需要的时候实现，计算梯度。

template <typename Dtype>

class Layer {

public:

/\*\*

\* 每个网络层需要自己定义它的setup而不需要构造函数

\*/

explicit Layer(const LayerParameter& param)

: layer\_param\_(param) {

//通过网络层参数来构造网络层

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));

}

}

}

// 析构函数

virtual ~Layer() {}

/\*\*

\* 实现一些通用的设置功能

\*

\* @param bottom 网络层的输入的shape

\* @param top 网络层的输出，需要被reshape

\* 调用 LayerSetUp 来对每一个网络层进行特殊化的处理,

\* 调用reshape top

\* 设置 数值权重

\* 这个方法可以不被重载。

\*/

void SetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {

CheckBlobCounts(bottom, top);

LayerSetUp(bottom, top);

Reshape(bottom, top);

SetLossWeights(top);

}

/\*\*

\* @brief 设置一些层相关的设置，定义的层需要实现这个方法以及Reshape方法

\*/

//设置网络层

virtual void LayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {}

/\*\*

\* @brief 调整top blob以适应bottom blob。

\*/

virtual void Reshape(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) = 0;

/\*\*

\* @brief 给定 bottom blobs, 计算 top blobs 以及 loss.

\* 每一个网络层都需要定义cpu版本的前馈，可选gpu版本的前馈

\*/

//前馈

inline Dtype Forward(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top);

/\*\*

\* @brief 给定 top blob 的梯度, 计算 bottom blob 梯度.

\* @param propagate\_down 向量，长度为ibottom 的个数，每个索引值表示是是否将损失梯度值反馈到该bottom中

\*/

//反馈

inline void Backward(const vector<Blob<Dtype>\*>& top,

const vector<bool>& propagate\_down,

const vector<Blob<Dtype>\*>& bottom);

/\*\*

\* @brief 返回可学习的参数 blobs.

\*/

vector<shared\_ptr<Blob<Dtype> > >& blobs() {

return blobs\_;

}

/\*\*

\* @brief 返回网络层参数

\*/

const LayerParameter& layer\_param() const { return layer\_param\_; }

//序列化

virtual void ToProto(LayerParameter\* param, bool write\_diff = false);

/\*\*

\* @brief 返回指定索引的标量损失值。

\*/

inline Dtype loss(const int top\_index) const {

return (loss\_.size() > top\_index) ? loss\_[top\_index] : Dtype(0);

}

/\*\*

\* @brief 设置网络层制定索引位置的loss

\*/

inline void set\_loss(const int top\_index, const Dtype value) {

if (loss\_.size() <= top\_index) {

loss\_.resize(top\_index + 1, Dtype(0));

}

loss\_[top\_index] = value;

}

/\*\*

\* @brief 返回网络层名字，字符串描述u

\*/

virtual inline const char\* type() const { return ""; }

//Bottom的blob的确切数目

virtual inline int ExactNumBottomBlobs() const { return -1; }

//Bottom blob的最小数目

virtual inline int MinBottomBlobs() const { return -1; }

//Botttom的确切数目

virtual inline int MaxBottomBlobs() const { return -1; }

//Top Blob的确切数目

virtual inline int ExactNumTopBlobs() const { return -1; }

//最小的blob的数目

virtual inline int MinTopBlobs() const { return -1; }

// 最大的blob的数目

virtual inline int MaxTopBlobs() const { return -1; }

// 是否bottom 和top的数目相同

virtual inline bool EqualNumBottomTopBlobs() const { return false; }

// 是否自动Top blob

virtual inline bool AutoTopBlobs() const { return false; }

//查询某一个bottom是否强制bp

virtual inline bool AllowForceBackward(const int bottom\_index) const {

return true;

}

//查询某一个blob是否bp

inline bool param\_propagate\_down(const int param\_id) {

return (param\_propagate\_down\_.size() > param\_id) ?

param\_propagate\_down\_[param\_id] : false;

}

//设置某一个blob是否bp。

inline void set\_param\_propagate\_down(const int param\_id, const bool value) {

if (param\_propagate\_down\_.size() <= param\_id) {

param\_propagate\_down\_.resize(param\_id + 1, true);

}

param\_propagate\_down\_[param\_id] = value;

}

protected:

// 网络层参数

LayerParameter layer\_param\_;

// 模式

Phase phase\_;

//用blob来存储一系列向量

vector<shared\_ptr<Blob<Dtype> > > blobs\_;

//是否bp的向量

vector<bool> param\_propagate\_down\_;

//存储top的loss

vector<Dtype> loss\_;

//cpu版本的前馈实现

virtual void Forward\_cpu(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) = 0;

//gpu版本的前馈实现

virtual void Forward\_gpu(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {

// LOG(WARNING) << "Using CPU code as backup.";

return Forward\_cpu(bottom, top);

}

//cpu版本的前馈实现

virtual void Backward\_cpu(const vector<Blob<Dtype>\*>& top,

const vector<bool>& propagate\_down,

const vector<Blob<Dtype>\*>& bottom) = 0;

//gpu版本的反馈实现

virtual void Backward\_gpu(const vector<Blob<Dtype>\*>& top,

const vector<bool>& propagate\_down,

const vector<Blob<Dtype>\*>& bottom) {

// LOG(WARNING) << "Using CPU code as backup.";

Backward\_cpu(top, propagate\_down, bottom);

}

// 核查bootom和top的大小是否与该layer层指定的一致。

virtual void CheckBlobCounts(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {

if (ExactNumBottomBlobs() >= 0) {

CHECK\_EQ(ExactNumBottomBlobs(), bottom.size())

<< type() << " Layer takes " << ExactNumBottomBlobs()

<< " bottom blob(s) as input.";

}

if (MinBottomBlobs() >= 0) {

CHECK\_LE(MinBottomBlobs(), bottom.size())

<< type() << " Layer takes at least " << MinBottomBlobs()

<< " bottom blob(s) as input.";

}

if (MaxBottomBlobs() >= 0) {

CHECK\_GE(MaxBottomBlobs(), bottom.size())

<< type() << " Layer takes at most " << MaxBottomBlobs()

<< " bottom blob(s) as input.";

}

if (ExactNumTopBlobs() >= 0) {

CHECK\_EQ(ExactNumTopBlobs(), top.size())

<< type() << " Layer produces " << ExactNumTopBlobs()

<< " top blob(s) as output.";

}

if (MinTopBlobs() >= 0) {

CHECK\_LE(MinTopBlobs(), top.size())

<< type() << " Layer produces at least " << MinTopBlobs()

<< " top blob(s) as output.";

}

if (MaxTopBlobs() >= 0) {

CHECK\_GE(MaxTopBlobs(), top.size())

<< type() << " Layer produces at most " << MaxTopBlobs()

<< " top blob(s) as output.";

}

if (EqualNumBottomTopBlobs()) {

CHECK\_EQ(bottom.size(), top.size())

<< type() << " Layer produces one top blob as output for each "

<< "bottom blob input.";

}

}

// 用blob初始化损失权重。

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

const int num\_loss\_weights = layer\_param\_.loss\_weight\_size();

if (num\_loss\_weights) {

CHECK\_EQ(top.size(), num\_loss\_weights) << "loss\_weight must be "

"unspecified or specified once per top blob.";

for (int top\_id = 0; top\_id < top.size(); ++top\_id) {

const Dtype loss\_weight = layer\_param\_.loss\_weight(top\_id);

if (loss\_weight == Dtype(0)) { continue; }

this->set\_loss(top\_id, loss\_weight);

const int count = top[top\_id]->count();

Dtype\* loss\_multiplier = top[top\_id]->mutable\_cpu\_diff();

caffe\_set(count, loss\_weight, loss\_multiplier);

}

}

}

DISABLE\_COPY\_AND\_ASSIGN(Layer);

}; // class Layer

// 前馈，根据caffe的mode 调用相对应的cpu实现或者是gpu实现，并且计算损失函数值。

template <typename Dtype>

inline Dtype Layer<Dtype>::Forward(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top) {

Dtype loss = 0;

Reshape(bottom, top);

switch (Caffe::mode()) {

case Caffe::CPU:

Forward\_cpu(bottom, top);

for (int top\_id = 0; top\_id < top.size(); ++top\_id) {

if (!this->loss(top\_id)) { continue; }

const int count = top[top\_id]->count();

const Dtype\* data = top[top\_id]->cpu\_data();

const Dtype\* loss\_weights = top[top\_id]->cpu\_diff();

loss += caffe\_cpu\_dot(count, data, loss\_weights);

}

break;

case Caffe::GPU:

Forward\_gpu(bottom, top);

#ifndef CPU\_ONLY

for (int top\_id = 0; top\_id < top.size(); ++top\_id) {

if (!this->loss(top\_id)) { continue; }

const int count = top[top\_id]->count();

const Dtype\* data = top[top\_id]->gpu\_data();

const Dtype\* loss\_weights = top[top\_id]->gpu\_diff();

Dtype blob\_loss = 0;

caffe\_gpu\_dot(count, data, loss\_weights, &blob\_loss);

loss += blob\_loss;

}

#endif

break;

default:

LOG(FATAL) << "Unknown caffe mode.";

}

return loss;

}

//反向传播梯度，根据Caffe的mode是在GPU还是CPU，调用相对应版本的函数

//propagate\_down 用于控制对应的层是否bp

template <typename Dtype>

inline void Layer<Dtype>::Backward(const vector<Blob<Dtype>\*>& top,

const vector<bool>& propagate\_down,

const vector<Blob<Dtype>\*>& bottom) {

switch (Caffe::mode()) {

case Caffe::CPU:

Backward\_cpu(top, propagate\_down, bottom);

break;

case Caffe::GPU:

Backward\_gpu(top, propagate\_down, bottom);

break;

default:

LOG(FATAL) << "Unknown caffe mode.";

}

}

// 序列化网络层参数到协议缓存，最终是调用blob写入协议缓存。

template <typename Dtype>

void Layer<Dtype>::ToProto(LayerParameter\* param, bool write\_diff) {

param->Clear();

param->CopyFrom(layer\_param\_);

param->clear\_blobs();

for (int i = 0; i < blobs\_.size(); ++i) {

blobs\_[i]->ToProto(param->add\_blobs(), write\_diff);

}

}

} // namespace caffe

#endif // CAFFE\_LAYER\_H\_

# [Caffe 中添加自己的网络层](http://blog.csdn.net/chenriwei2/article/details/46432727)

分类： [caffe](http://blog.csdn.net/chenriwei2/article/category/3146215) [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319)2015-06-09 22:38 387人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/46432727#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/46432727#report)

目录[(?)[+]](http://blog.csdn.net/chenriwei2/article/details/46432727)

### 写在前面：

Caffe 中有众多的网络层，最新版本的代码已经涵盖了很多种类型的网络层，然而，有时候由于各种原因，其给定的网络层不能满足我们的要求，这时候就要对其更改，以使其满足自己的需求，感谢作者开源代码以及众多的代码维护者。

由于Caffe 中的网络层都是直接或者间接地给予Layer 基类，所以，在我们需要添加新的类型时，就需要选择好自己的基类，以使我们能够更好的利用基类已有的一些方法。我们新建的类可以基于   
1. 直接继承于Layer   
2. 继承于DataLayer   
3. 继承于NeuronLayer   
4. 继承于LossLayer   
5. 或者如果是直接想改进某个层的代码，可以直接继承该类，并做相对应的修改即可。

### 具体步骤：

第一步， 添加头文件.h ,可以自己新建一个头文件，或者追加到一个已有的头文件后面，比如include/DataLayer.hpp 等, 在里面定义一些类变量等。

第二步，添加实现文件，创建类的实现函数，放在src/XXX.cpp 中，如果有cuda实现，则需要添加相对应的.cu文件实现，主要需要写LayerSetup ，reshape ，forward ，backup 函数。

第三步， 注册类，到src/caffe/layer\_factory.cpp 中添加对应的类，在其中加入

REGISTER\_LAYER\_CLASS(MYLAYER, MyLayer);

* 1

第四步，在src/caffe/proto/caffe.proto 文件中，添加对应的类以及类参数。   
在enum LayerType {}中添加一行

enum LayerType {

MYLAYER = ID

}

//其中MYLAYER 是新添加的类名，ID是类编号，注意不能更之前的重复。

* 1
* 2
* 3
* 4

再添加类参数

optional MylayerParameter mylayer\_param = ID2;

//其中MylayerParameter 是网络层参数，ID2 是ID

* 1
* 2

最后定义类参数

message MylayerParameter {

optional uint32 myparam1 = 1;

optional bool myparam2 = 2 [default = true];

}

* 1
* 2
* 3
* 4

第五步，写测试文件，这个是可选的，不过为了合并到github中，需要测试通过。

# [affe中的损失函数解析](http://blog.csdn.net/chenriwei2/article/details/45291739)

分类： [深度学习](http://blog.csdn.net/chenriwei2/article/category/2339319) [caffe](http://blog.csdn.net/chenriwei2/article/category/3146215)2015-04-26 20:33 1002人阅读 [评论](http://blog.csdn.net/chenriwei2/article/details/45291739#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenriwei2/article/details/45291739#report)

目录[(?)[+]](http://blog.csdn.net/chenriwei2/article/details/45291739)

# Caffe中的损失函数解析

## 导言

在有监督的机器学习中，需要有标签数据，与此同时，也需要有对应的损失函数（Loss Function）。

在Caffe中，目前已经实现了一些损失函数，包括最常见的L2损失函数，对比损失函数，信息增益损失函数等等。在这里做一个笔记，归纳总结Caffe中用到的不同的损失函数，以及分析它们各自适合的使用场景。

## 欧式距离损失函数（Euclidean Loss）

**输入：**

预测的值： y^∈[−∞,+∞], 其中，它们的形状为：N×C×H×W

标签的值： y∈[−∞,+∞], 其中，它们的形状为：N×C×H×W

**输出：**

损失的值：Loss=12N∑Nn=1∥y^n−yn∥22

**适合场景：**

回归，特别是其回归的值是实数值得时候。

## 对比损失函数（Contrastive loss）

**输入：**

形状：(N×C×1×1) 特征 a∈[−∞,+∞]

形状：(N×C×1×1) 特征 b∈[−∞,+∞]

形状：(N×1×1×1) 相似性 y∈[0,1]

**输出：**

形状：(1×1×1×1)

对比损失函数为: E=12N∑n=1N(y)d+(1−y)max(margin−d,0)

其中 d=||an−bn||22.

**适合场景：**

可以用来训练Siamese网络

## 铰链损失函数（Hinge Loss）

**输入：**

形状：(N×C×H×W) 预测值 t∈[−∞,+∞] 代表着预测 K=CHW 个类中的得分（注：CHW表示着在网络设计中，不一定要把预测值进行向量化，只有其拉直后元素的个数相同即可。） . 在SVM中, t 是 D 维特征X∈RD×N, 和学习到的超平面参数W∈RD×K 内积的结果 XTW   
所以，一个网络如果仅仅只有全连接层 + 铰链损失函数，而没有其它的可学习的参数，那么它就等价于SVM

标签值：

(N×1×1×1) 标签 l, 是一个整数类型的数 ln∈[0,1,2,...,K−1] 其代表在 K 个类中的正确的标签。

**输出：**

形状：(1×1×1×1)   
损失函数计算: E=1N∑n=1N∑k=1K[max(0,1−δ{ln=k}tnk)]p, Lp 范数 (默认是 p=1, 是 L1 范数; L2 范数,正如在 L2-SVM中一样,也有实现),

其中 δ{条件}={1−1成立不成立

**应用场景：**

在一对多的分类中应用,类似于SVM.

## 信息增益损失函数（InformationGain Loss）

**输入：**

1. 形状：(N×C×H×W) 预测值 p^∈[0,1] 内， 表示这预测每一类的概率，共 K=CHW 个类， 每一个预测 概率p^n 的和为1: ∀n∑k=1Kp^nk=1.
2. 形状：(N×1×1×1) 标签值： l, 是一个整数值，其范围是 ln∈[0,1,2,...,K−1] 表示着在 K 个类中的索引。
3. 形状：(1×1×K×K) (可选) 信息增益矩阵 H.作为第三个输入参数，. 如果 H=I, 则它等价于多项式逻辑损失函数

**输出：**

形状：(1×1×1×1)

计算公式: E=−1N∑n=1NHlnlog(p^n)=−1N∑n=1N∑k=1KHln,klog(p^n,k), 其中 Hln 表示 行 ln of H.

## 多项式逻辑损失函数（Multinomial Logistic Loss）

**输入：**

形状：(N×C×H×W) 预测值 p^∈[0,1]范围中， 表示这预测的每一类的概率，共 K=CHW 个类. 每一个预测概率p^n 的和为1: ∀n∑k=1Kp^nk=1.

形状：(N×1×1×1) 标签 l, 是一个整数值，其范围是 ln∈[0,1,2,...,K−1] 表示着在 K 个类中的索引。

**输出：**

形状：(1×1×1×1) 计算公式: E=−1N∑n=1Nlog(p^n,ln)

**应用场景：**

在一对多的分类任务中使用，直接把预测的概率分布作为输入.

## Sigmoid 交叉熵损失函数（Sigmoid Cross Entropy Loss）

**输入：**

1. 形状: (N×C×H×W) 得分 x∈[−∞,+∞], 这个层使用 sigmoid 函数 σ(.) 映射到概率分布 p^n=σ(xn)∈[0,1]
2. 形状:(N×C×H×W) 标签 y∈[0,1]

**输出：**

1. 形状：(1×1×1×1) 计算公式: E=−1n∑n=1N[pnlogp^n+(1−pn)log(1−p^n)]

**应用场景：**   
预测目标概率分布

## Softmax+损失函数(Softmax With Loss)

**输入：**

1. 形状：(N×C×H×W) 预测值 x∈[−∞,+∞] 代表预测每个类的得分。 共 K=CHW 类. 这一层把得分通过softmax映射到概率分布 p^nk=exp(xnk)/[∑k′exp(xnk′)]
2. 形状：(N×1×1×1) 标签值 是一个整数值，其范围是 ln∈[0,1,2,...,K−1] 表示着在 K 个类中的索引。

**输出：**

1. 形状：(1×1×1×1) 计算公式: E=−1N∑n=1Nlog(p^n,ln), 其中 p^ 为softmax输出的类概率。

**应用场景：**

在一对多分类中应用。