# [caffe源码解析 — train\_net.cpp](http://blog.csdn.net/qq_16055159/article/details/45056129)

train\_net.cpp是caffe的主函数所在处，学习caffe时应该由此辐射到各个具体的函数去学习。

1. 调用ReadProtoFromTextFileOrDie()从argv[1]中读入solver\_param：   
   SolverParameter solver\_param
2. 用solver\_param去初始化solver，solverSGDSolver：   
   solver(solver\_param)
3. 调用solver的Solve函数，solver.Solve(argv[2]) 或 solver.Solve()

**train\_net.cpp**

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

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

::google::SetLogDestination(0, "../tmp/");

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

LOG(ERROR) << "Usage: train\_net solver\_proto\_file [resume\_point\_file]";

return 1;

}

SolverParameter solver\_param;

ReadProtoFromTextFileOrDie(argv[1], &solver\_param);

LOG(INFO) << "Starting Optimization";

SGDSolver<float> solver(solver\_param);

if (argc == 3) {

LOG(INFO) << "Resuming from " << argv[2];

solver.Solve(argv[2]);

} else {

solver.Solve();

}

LOG(INFO) << "Optimization Done.";

return 0;

}

# [caffe源码解析 — blob.cpp](http://blog.csdn.net/qq_16055159/article/details/45099547)

分类： [caffe源码](http://blog.csdn.net/qq_16055159/article/category/3107705)2015-04-17 17:17 269人阅读 [评论](http://blog.csdn.net/qq_16055159/article/details/45099547#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qq_16055159/article/details/45099547#report)

[源码](http://www.csdn.net/tag/%e6%ba%90%e7%a0%81)

[主要参考：linger](http://blog.csdn.net/lingerlanlan/article/details/24379689)

**Reshape(const int num, const int channels, const int height, const int width)**   
功能：改变一个blob的大小   
步骤：1.读入num\_，channels\_，height\_，width\_的大小   
2.计算count\_：count\_ = num\_ \* channels\_ \* height\_ \* width\_;   
3.如果count\_不为0，则重新为data\_和diff\_分配一块空间   
如果count为0，则都初始化为NULL   
输入：num，channels，height，width   
输出：无

**ReshapeLike(const Blob<Dtype>& other)**   
功能：为data\_和diff\_ 重新分配一块空间，大小和另一个blob的一样   
输入：Bolb类型的other   
输出：无

**Blob<Dtype>::Blob(const int num, const int channels, const int height, const int width)**   
功能：简单的构造函数   
输入：num，channels，height，width

**set\_cpu\_data(Dtype\* data)**   
功能：改变CPU的数据

const Dtype\* Blob<Dtype>::cpu\_data()

const Dtype\* Blob<Dtype>::cpu\_diff()

const Dtype\* Blob<Dtype>::gpu\_data()

const Dtype\* Blob<Dtype>::gpu\_diff()

* 1
* 2
* 3
* 4

功能：以上四个函数，前两个调用to\_cpu(),返回cpu\_ptr；第一个对于data对象，第二个对于diff对象   
后两个 调用to\_gpu(),返回gpu\_ptr；第一个对于data对象，第二个对于diff对象

Dtype\* Blob<Dtype>::mutable\_cpu\_data()

Dtype\* Blob<Dtype>::mutable\_gpu\_data()

Dtype\* Blob<Dtype>::mutable\_cpu\_diff()

Dtype\* Blob<Dtype>::mutable\_gpu\_diff()

* 1
* 2
* 3
* 4

功能：下面是gpu\_data和mutable\_gpu\_data函数的对比，可以看出第二个函数   
只是多改变了head\_的值。

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

to\_gpu();

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

}

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

to\_gpu();

head\_ = HEAD\_AT\_GPU;

return gpu\_ptr\_;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

void Blob<Dtype>::ShareData(const Blob& other)

void Blob<Dtype>::ShareDiff(const Blob& other)

* 1
* 2

功能：从other的blob复制data和diff的值

**Update()**   
功能：更新data\_的数据，就是减去diff\_的数据。   
步骤：1.判断blob的位置（HEAD\_AT\_CPU/HEAD\_AT\_GPU/SYNCED/UNINITIALIZED）   
1）调用caffe\_axpy：在math\_functions.cpp可以找到该函数的实现，其实这函数也是封装了mkl的函数。这里调用是为了实现了两个向量的减法。   
2）调用caffe\_gpu\_axpy：在math\_functions.cpp可以找到该函数的实现，其实这函数也是封装了cublas的函数。这里调用是为了实现了两个向量的减法。

**CopyFrom(const Blob& source, bool copy\_diff, bool reshape)**   
功能：从source拷贝数据。copy\_diff作为标志来区分是拷贝data还是拷贝diff。   
步骤：1.如果是GPU：   
如果是拷贝diff：调用cudaMemcpy函数将source的diff拷贝过来   
否则拷贝data   
2.如果是CPU：   
如果是拷贝diff：调用memcpy函数将source的diff拷贝过来   
否则拷贝data

**FromProto(const BlobProto& proto)**   
功能：从proto读数据进来，其实就是反序列化   
步骤：1.先把blob的大小改变一下   
2.得到cpu中数据的地址   
3.用proto中的data覆盖blob中的data   
4.用proto中的diff覆盖blob中的diff

**ToProto(BlobProto\* proto, bool write\_diff)**   
功能：把blob数据保存到proto中

# [caffe源码解析 — SyncedMemory.cpp](http://blog.csdn.net/qq_16055159/article/details/45099591)

分类： [caffe源码](http://blog.csdn.net/qq_16055159/article/category/3107705)2015-04-17 17:20 155人阅读 [评论](http://blog.csdn.net/qq_16055159/article/details/45099591#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qq_16055159/article/details/45099591#report)

[**主要参考：linger**](http://blog.csdn.net/lingerlanlan/article/details/24379689)

**SyncedMemory::~SyncedMemory()**   
功能：析构函数   
步骤：1.如果有CPU数据则释放   
2.如果有GPU数据则释放

**inline void SyncedMemory::to\_cpu()**   
功能：把数据放到cpu上   
1.数据未初始化，则在cpu申请内存（申请为0）。此时状态为HEAD\_AT\_CPU   
2.数据本来在gpu，则从gpu拷贝内存到cpu。此时状态为SYNCED   
3.数据本来在cpu，不做处理   
4.数据在cpu和gpu都有，不做处理

**inline void SyncedMemory::to\_gpu()**   
功能：把数据放到gpu上   
1.数据未初始化，在gpu申请内存（申请为0）。此时状态为HEAD\_AT\_GPU   
2.数据在cpu，从cpu拷贝到gpu。此时状态为SYNCED   
3.数据在gpu，不做操作。   
4.数据在cpu和gpu都有，不做操作。

**const void\* SyncedMemory::cpu\_data()**   
功能：返回数据在cpu的指针

**void SyncedMemory::set\_cpu\_data(void\* data)**   
功能：清空CPU的数据

**void\* mutable\_cpu\_data()**   
功能：返回数据在cpu的指针，并改变数据的状态为HEAD\_AT\_CPU

**void\* mutable\_gpu\_data()**   
功能：返回数据在cpu的指针，并改变数据的状态为HEAD\_AT\_GPU

# [caffe源码解析 — net.cpp](http://blog.csdn.net/qq_16055159/article/details/45057297)

分类： [caffe源码](http://blog.csdn.net/qq_16055159/article/category/3107705)2015-04-15 11:52 590人阅读 [评论](http://blog.csdn.net/qq_16055159/article/details/45057297#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/qq_16055159/article/details/45057297#report)

Net类是Solve类的一个成员，在net.cpp中定义了对Net的所有操作，其中包括：

* Init
* GetLearningRateAndWeightDecay
* ForwardPrefilled
* Backward
* ShareTrainedLayersWith
* CopyTrainedLayersFrom
* ToProto
* Update
* has\_blob
* blob\_by\_name
* has\_layer
* layer\_by\_name

**Net(const NetParameter& param)**   
功能：调用Init函数初始化网络   
输入：NetParameter& param   
输出：无

**Net(const string& param\_file)**   
功能：调用Init函数初始化网络   
输入：string& param\_file   
输出：无

**Init(const NetParameter& in\_param)**   
功能：初始化网络   
输入：NetParameter& in\_param   
输出：无   
步骤：   
**<1>** 调用InsertSplits()函数从in\_param读入新网络到param   
**<2>** 定义name\_，blob\_name\_to\_idx，available\_blobs，num\_layers   
**<3>** param.input\_size()返回输入层blob的个数;   
param.input(i)表示第i个blob的名字;   
param.layers\_size()返回网络的层数。   
**<4>** 对每一个输入层的blob：

1. 产生一块和当前blob一样大的空间 e.g. imput\_dim=[12 55 66 39 20 24 48 64]表示第一个blob的四个维数为 12 55 66 39，第二个为 20 24 48 64 接着blob\_pointer指向这块空间
2. blob\_pointer压到blobs\_中 vector<shared\_ptr<Blob<Dtype>>> blobs\_
3. blob\_name压到blob\_names\_中 vector<string> blob\_names\_
4. param.force\_backward()压到blob\_need\_backward\_中   
   vector<bool> blob\_need\_backward\_
5. i 压到 net\_input\_blob\_indices\_中 net\_input\_blob\_indices\_ -> vector
6. blob\_pointer.get() 压到 net\_input\_blobs\_中   
   注意与blobs\_的区别   
   vector<shared\_ptr<Blob<Dtype>>> blobs\_   
   vector<Blob<Dtype>\*> net\_input\_blobs\_   
   shared\_ptr类型的参数调用.get()则得到Blob\*类型
7. map<string, int> blob\_name\_to\_idx
8. 初始化为输入层的每个blob的名字 set<string> available\_blobs
9. 计算所需内存 memory\_used += blob\_pointer->count()

**<5>** 存每一层的输入blob指针 vector<vector<Blob<Dtype>\*> > bottom\_vecs\_   
存每一层输入(bottom)的id vector<vector<int> > bottom\_id\_vecs\_   
存每一层输出(top)的blob vector<vector<Blob<Dtype>\*> > top\_vecs\_   
用网络的层数param.layers\_size()去初始化上面四个变量   
vector<vector<int> > top\_id\_vecs\_   
**<6>** 对第i层（很大的一个for循环）：

1. param.layers(i)返回的是关于第当前层的参数：   
   layer\_param = param.layers(i)
2. 把当前层的参数转换为shared\_ptr<Layer<Dtype>>，并压入到layers\_中
3. 把当前层的名字压入到layer\_names\_：vector<string> layer\_names\_
4. 判断当前层是否需要反馈 need\_backward = param.force\_backward()
5. 下面开始产生当前层：分为处理bottom的blob和top的blob两个步骤   
   **对第j个bottom的blob：**
   * layer\_param.bottom\_size()存的是当前层的输入blob数量
   * layer\_param.bottom(j)存的是第j个输入blob的名字
   * 读取当前blob的id，**其中blob\_name\_to\_idx在输入层初始化过了**   
     blob\_name\_to\_idx[blob\_name] = i
   * 输出当前blob的名字
   * 存入第j个输入blob的指针bottom\_vecs\_[i].push\_back(blobs\_[blob\_id].get())
   * 存入第j个输入blob的id bottom\_id\_vecs\_[i].push\_back(blob\_id)
   * 更新need\_backward
   * 从available\_blobs中删除第j个blob的名字

**对第j个top的blob：**

* + layer\_param.top\_size()存的是当前层的输出blob数量
  + layer\_param.top(j)存的是第j个输出blob的名字
  + 判断是否进行同址计算
  + 输出当前blob的名字
  + 定义一块新的blob空间，用blob\_pointer指向这块空间
  + 把这个指针存入到blobs\_中
  + 把blob\_name、force\_backward、idx存入对应的容器中
  + 向available\_blobs插入当前blob的名字
  + top\_vecs\_[i]对于第i层，插入当前blob的指针
  + top\_id\_vecs\_[i]对于第i层，插入当前blob的id

1. 输出当前层位于top的blob的信息
2. 计算所需内存
3. 判断当前层i是否需要backward

**<7>** 所有名字在available\_blobs中的blob为当前层的输出blob，   
存入net\_output\_blobs\_中   
**<8>** 建立每个blob的name和index的对应关系map：blob\_names\_index\_   
**<9>** 建立每个层的name和index的对应关系map：layer\_names\_index\_   
**<10>** 调用GetLearningRateAndWeightDecay函数

**GetLearningRateAndWeightDecay()**   
功能：收集学习速率和权重衰减，即更新params\_、params\_lr\_和params\_weight\_decay\_   
输入：无   
输出：无   
步骤：对每一层   
1. 把当前层的所有blob存入params\_中   
params\_// The parameters in the network   
2. 如果有lr, 则把当前层的所有blob的lr存入params\_lr\_中; 否则, lr默认为1   
3. 如果有 weight\_decay，则把当前层的所有 blob 的 weight\_decay   
存入 params\_weight\_decay\_ 中   
4. 否则，weight\_decay 默认为1

**ForwardPrefilled(Dtype\* loss)**   
功能：前馈预先填满，即预先进行一次前馈   
输入：Dtype\* loss   
输出：net\_output\_blobs\_，前馈后的输出层blob：vector

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

{

switch (Caffe::mode())

{

case Caffe::CPU:

return Forward\_cpu(bottom, top);

case Caffe::GPU:

return Forward\_gpu(bottom, top);

default:

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

return Dtype(0);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

virtual Dtype Forward\_cpu(const vector<Blob<Dtype>\*>& bottom, vector<Blob<Dtype>\*>\* top) = 0;

}

* 1
* 2

**Forward(const vector<Blob<Dtype>\*> & bottom, Dtype\* loss)**   
功能：把网络输入层的blob读到net\_input\_blobs\_，然后进行前馈，计算出loss   
输入：整个网络输入层的blob   
输出：整个网络输出层的blob

**Forward(const string& input\_blob\_protos, Dtype\* loss)**   
功能：Forward的重载，只是输入层的blob以string的格式传入

**Backward()**   
功能：对整个网络进行反向传播

**ShareTrainedLayersWith(Net\* other)**   
功能：从Other网络复制某些层   
步骤：对Other网络的第i层（源层）：   
1. 定义一个Layer的指针指向第i层   
2. 读取第i层（源层）的名字   
3. 找通过名字来找目标层   
如果没找到，即target\_layer\_id == layer\_names\_.size()   
则忽略Other的第i层，即Other的第i层不需要share给网络   
4. 如果找到了，即other的第i层需要share给网络，   
则把目标层的所有blob读到target\_blobs中

1. 判断目标层和源层的blob数量是否相等
2. 判断每个blob大小是否相等
3. 调用ShareData函数把源层的blob赋给目标层的blob

**CopyTrainedLayersFrom(const NetParameter& param)**   
功能：和ShareTrainedLayersWith一样   
步骤：不同的是调用FromProto函数把源层的blob赋给目标层的blob

**CopyTrainedLayersFrom(const string trained\_filename)**   
功能：从文件中读入NetParameter param，然后调用CopyTrainedLayersFrom()

**ToProto(NetParameter\* param, bool write\_diff)**   
功能：把网络的参数存入prototxt中   
步骤：   
1. 设置网络的名字：param->set\_name(name\_)   
2. 加入输入层blob的名字   
3. 对于第i层：

1. 加入bottom的blob的名字
2. 加入top的blob的名字
3. 写到proto中

**Update()**   
功能：更新params\_中blob的值

**has\_blob(const string& blob\_name)**   
功能：判断是否存在名字为blob\_name的blob

**blob\_by\_name(const string& blob\_name)**   
功能：给一个blob的名字，返回这个blob的指针

**has\_layer(const string& layer\_name)**   
功能：判断是否存在名字为layer\_name的layer

**layer\_by\_name(const string& layer\_name)**   
功能：给一个layer的名字，返回这个layer的指针

# [caffe源码解析 — solver.cpp](http://blog.csdn.net/qq_16055159/article/details/45068147)

**Solver<Dtype>::Solver(const SolverParameter& param)**   
功能：构造函数   
步骤：初始化两个Net类，net\_和test\_net\_，并调用Init()函数   
输入：SolverParameter类型的param   
输出：无

**Solver<Dtype>::Solver(const string& param\_file)**   
功能：构造函数   
步骤：初始化两个Net类，net\_和test\_net\_，并调用Init()函数   
输入：string类型的param\_file   
输出：无

**void Solver<Dtype>::Init(const SolverParameter& param)**   
功能：初始化网络   
步骤：   
1. 设置随机数种子   
2. 申请一块Net空间以下面的构造函数进行初始化   
param\_file=train\_net\_，net\_指向这块空间   
3. 如果有test\_net，则申请一块Net空间，test\_net\_指向这块空间   
输入：SolverParameter类型的param   
输出：无

Net<Dtype>::Net(const string& param\_file) {

NetParameter param;

ReadNetParamsFromTextFileOrDie(param\_file, &param);

Init(param);

}

* 1
* 2
* 3
* 4
* 5

**void Solver<Dtype>::Solve(const char\* resume\_file)**   
功能：训练网络   
步骤：   
1. 设置Caffe的mode（GPU还是CPU）   
2. 如果是GPU且有GPU芯片的ID，则设置GPU   
3. 设置当前阶段（TRAIN还是TEST/TRAIN）   
4. 调用PreSolve函数：PreSolve()   
5. 调用Restore函数：Restore(resume\_file)   
6. 调用一遍Test()，判断内存是否够   
7. 对于每一次训练时的迭代(遍历整个网络):while (iter\_++ < param\_.max\_iter())

1. 计算loss：loss = net\_->ForwardBackward(bottom\_vec)其中：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ForwardBackward() \*\*\*\*\*\*\*\*\*\*\*\*

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

Dtype loss;

Forward(bottom, &loss);

Backward();

return loss;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Forward() \*\*\*\*\*\*\*\*\*\*\*

const vector<Blob<Dtype>\*>& Net<Dtype>::Forward(

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

// Copy bottom to internal bottom

for (int i = 0; i < bottom.size(); ++i)

net\_input\_blobs\_[i]->CopyFrom(\*bottom[i]){;

}

return ForwardPrefilled(loss);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ForwardPrefilled() \*\*\*\*\*\*\*\*\*\*\*\*

const vector<Blob<Dtype>\*>& Net<Dtype>::ForwardPrefilled(Dtype\* loss) {

if (loss != NULL) {

\*loss = Dtype(0.);

}

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

// LOG(ERROR) << "Forwarding " << layer\_names\_[i];

Dtype layer\_loss = layers\_[i]->Forward(bottom\_vecs\_[i], &top\_vecs\_[i]);

if (loss != NULL) {

\*loss += layer\_loss;//对于非loss层都会返回0：return Dtype(0.);

}

}

return net\_output\_blobs\_;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Layer::Forward() \*\*\*\*\*\*\*\*\*\*\*\*

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

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

switch (Caffe::mode()) {

case Caffe::CPU:

return Forward\_cpu(bottom, top);//虚函数，不同层有不同层的计算方法

case Caffe::GPU:

return Forward\_gpu(bottom, top);

default:

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

return Dtype(0);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Backward() \*\*\*\*\*\*\*\*\*\*\*\*

void Net<Dtype>::Backward() {

for (int i = layers\_.size() - 1; i >= 0; --i) {

if (layer\_need\_backward\_[i]) {

layers\_[i]->Backward(top\_vecs\_[i], true, &bottom\_vecs\_[i]);

}

}

}

* 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

2.调用ComputeUpdateValue函数：ComputeUpdateValue()   
3. 输出loss   
4. 达到test\_interval时调用Test()   
5. 达到snapshot时调用snapshot()   
6. 调用Snapshot函数：Snapshot()   
输入：char\*类型的resume\_file   
输出：无

**void Solver<Dtype>::Test()**   
功能：测试网络   
输入：无   
输出：无   
步骤：   
1. 设置当前阶段（TRAIN还是TEST/TEST）   
2. 将test\_net\_指向net\_，即对同一个网络操作   
3. 对于每一次测试时的迭代：for (int i = 0; i < param\_.test\_iter(); ++i)

1. 用下面语句给result赋值net\_output\_blobs\_ //result是所有的输出层blob   
   同时得到这次测试的iter\_loss   
   **result = test\_net\_->Forward(bottom\_vec, &iter\_loss)**
2. 第一次测试时：   
   1. 取每一个输出层的blob result\_vec = result[j]->cpu\_data()
   2. 把每一个blob的数据（降为一维）存入一个vector–“test\_score”
3. 不是第一次测试：   
   1. 用 test\_score[idx++] += result\_vec[k]   
      而不是 test\_score.push\_back(result\_vec[k])
   2. 把输出层对应位置的blob值累加   
      test\_score[idx++] += result\_vec[k]
4. 是否要输出Test loss
5. 是否要输出test\_score
6. 设置当前阶段（TRAIN还是TEST/TRAIN）

**void Solver<Dtype>::Snapshot()**   
功能：输出当前网络状态到一个文件中，不重要   
输入：无   
输出：无

**void Solver<Dtype>::Restore(const char\* state\_file)**   
功能：从一个文件中读入网络状态，并可以从那个状态恢复，不重要   
输入：文件名   
输出：无

**Dtype SGDSolver<Dtype>::GetLearningRate()**   
功能：得到学习率   
步骤：   
1. 得到学习率类型 const string& lr\_policy = this->param\_.lr\_policy()   
2. 判断学习率类型（注释有介绍）   
3. 返回学习率   
输入：无   
输出：Dtype类型的rate

**void SGDSolver<Dtype>::PreSolve()**   
功能：提前训练   
步骤：   
1. 将训练网络net\_的参数读到net\_params net\_params = this->net\_->params()   
其中params\_是一个存blob指针的vector   
2. 清空历史残留值   
3. 向history压入与网络的每一层blob相同大小的空间   
输入：无   
输出：无

**void SGDSolver<Dtype>::ComputeUpdateValue()**   
功能：用随机梯度下降法计算更新值   
输入：无   
输出：无   
步骤：   
1. (所有的)读取网络参数net\_params，网络学习速率 net\_params\_lr，   
权值衰减net\_params\_weight\_decay 读取学习速率rate   
2. (当前层)读取动量，权值衰减   
3. 如果是CPU：   
对于每一次层：

1. 计算local\_rate，local\_decay
2. 调用caffe\_cpu\_axpby，caffe\_axpy，caffe\_copy函数：

caffe\_cpu\_axpby(net\_params[param\_id]->count(), local\_rate, net\_params[param\_id]->cpu\_diff(), momentum, history\_[param\_id]->mutable\_cpu\_data());

* 1

caffe\_axpy(net\_params[param\_id]->count(), local\_decay\*local\_rate, net\_params[param\_id]->cpu\_data(),history\_[param\_id]->mutable\_cpu\_data());

* 1

void caffe\_cpu\_axpby<float>(const int N, const float alpha, const float\* X,const float beta, float\* Y)

{

cblas\_saxpby(N, alpha, X, 1, beta, Y, 1);

}

其中:

inline void cblas\_saxpby(const int N, const float alpha, const float\* X,const int incX, const float beta, float\* Y, const int incY)

{

cblas\_sscal(N, beta, Y, incY);

cblas\_saxpy(N, alpha, X, incX, Y, incY);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

caffe\_cpu\_axpby调用了cblas\_saxpby，即调用了cblas\_sscal和cblas\_saxpy

void caffe\_axpy<float>(const int N, const float alpha, const float\* X,float\* Y)

{

cblas\_saxpy(N, alpha, X, 1, Y, 1);

}

* 1
* 2
* 3
* 4

caffe\_axpy调用了cblas\_saxpy，即调用了cblas\_saxpy   
所以caffe\_cpu\_axpby比caffe\_axpy多输入了一个beta参数，多调用了cblas\_sscal(N, beta, Y, incY);   
4. GPU同理

**void SGDSolver<Dtype>::SnapshotSolverState(SolverState\* state)**   
略

**void SGDSolver<Dtype>::RestoreSolverState(const SolverState& state)**   
略

# [caffe源码解析 — caffe.proto](http://blog.csdn.net/qq_16055159/article/details/45115359)

## 引言

要看caffe源码，我认为**首先**应该看的就是**caffe.proto**。   
它位于…\src\caffe\proto目录下，在这个文件夹下还有一个.pb.cc和一个.pb.h文件，这两个文件都是由caffe.proto编译而来的。   
在caffe.proto中定义了很多结构化数据，包括：

* BlobProto
* Datum
* FillerParameter
* NetParameter
* SolverParameter
* SolverState
* LayerParameter
* ConcatParameter
* ConvolutionParameter
* DataParameter
* DropoutParameter
* HDF5DataParameter
* HDF5OutputParameter
* ImageDataParameter
* InfogainLossParameter
* InnerProductParameter
* LRNParameter
* MemoryDataParameter
* PoolingParameter
* PowerParameter
* WindowDataParameter
* V0LayerParameter

## 正文

### ****一、什么是protocol buffer****

**以下内容摘自：**[Google Protocol Buffer 的使用和原理](http://www.ibm.com/developerworks/cn/linux/l-cn-gpb/)   
**强烈推荐另外一篇极好的博文是：**[Protocol Buffer技术详解(C++实例)](http://www.cnblogs.com/stephen-liu74/archive/2013/01/04/2842533.html)

### ****简介****

什么是 Google Protocol Buffer？ 假如您在网上搜索，应该会得到类似这样的文字介绍：   
Google Protocol Buffer( 简称 Protobuf) 是 Google 公司内部的混合语言数据标准，目前已经正在使用的有超过 48,162 种报文格式定义和超过 12,183 个 .proto 文件。他们用于 RPC 系统和持续数据存储系统。   
Protocol Buffers 是一种轻便高效的结构化数据存储格式，可以用于结构化数据串行化，或者说序列化。它很适合做数据存储或 RPC 数据交换格式。可用于通讯协议、数据存储等领域的语言无关、平台无关、可扩展的序列化结构数据格式。目前提供了 C++、Java、Python 三种语言的 API。   
或许您和我一样，在第一次看完这些介绍后还是不明白 Protobuf 究竟是什么，那么我想一个简单的例子应该比较有助于理解它。

### ****一个简单的例子****

安装 Google Protocol Buffer   
在网站 <http://code.google.com/p/protobuf/downloads/list>上可以下载 Protobuf 的源代码。然后解压编译安装便可以使用它了。   
安装步骤如下所示：

tar -xzf protobuf-2.1.0.tar.gz

cd protobuf-2.1.0

./configure --prefix=$INSTALL\_DIR

make

make check

make install

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

### ****关于简单例子的描述****

我打算使用 Protobuf 和 C++ 开发一个十分简单的例子程序。   
该程序由两部分组成。第一部分被称为 Writer，第二部分叫做 Reader。   
Writer 负责将一些结构化的数据写入一个磁盘文件，Reader 则负责从该磁盘文件中读取结构化数据并打印到屏幕上。   
准备用于演示的结构化数据是 HelloWorld，它包含两个基本数据：

* ID，为一个整数类型的数据
* Str，这是一个字符串

### ****书写 .proto 文件****

首先我们需要编写一个 proto 文件，定义我们程序中需要处理的结构化数据，在 protobuf 的术语中，结构化数据被称为 Message。proto 文件非常类似 java 或者 C 语言的数据定义。代码清单 1 显示了例子应用中的 proto 文件内容。   
清单 1. proto 文件

package lm;

message helloworld

{

required int32 id = 1; // ID

required string str = 2; // str

optional int32 opt = 3; //optional field

}

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

一个比较好的习惯是认真对待 proto 文件的文件名。比如将命名规则定于   
packageName.MessageName.proto   
在上例中，package 名字叫做 lm，定义了一个消息 helloworld，该消息有三个成员，类型为 int32 的 id，另一个为类型为 string 的成员 str。opt 是一个可选的成员，即消息中可以不包含该成员。

### ****编译 .proto 文件****

写好 proto 文件之后就可以用 Protobuf 编译器将该文件编译成目标语言了。本例中我们将使用 C++。   
假设您的 proto 文件存放在 $SRC\_DIR 下面，您也想把生成的文件放在同一个目录下，则可以使用如下命令：

protoc -I=$SRC\_DIR --cpp\_out=$DST\_DIR $SRC\_DIR/addressbook.proto

* 1

命令将生成两个文件：   
lm.helloworld.pb.h ， 定义了 C++ 类的头文件   
lm.helloworld.pb.cc ， C++ 类的实现文件   
在生成的头文件中，定义了一个 C++ 类 helloworld，后面的 Writer 和 Reader 将使用这个类来对消息进行操作。诸如对消息的成员进行赋值，将消息序列化等等都有相应的方法。

### ****编写 writer 和 Reader****

如前所述，Writer将把一个结构化数据写入磁盘，以便其他人来读取。假如我们不使用 Protobuf，其实也有许多的选择。一个可能的方法是将数据转换为字符串，然后将字符串写入磁盘。转换为字符串的方法可以使用sprintf()，这非常简单。数字123可以变成字符串“123”。   
这样做似乎没有什么不妥，但是仔细考虑一下就会发现，这样的做法对写 Reader 的那个人的要求比较高，Reader 的作者必须了 Writer 的细节。比如”123”可以是单个数字 123，但也可以是三个数字 1,2 和 3，等等。这么说来，我们还必须让 Writer 定义一种分隔符一样的字符，以便 Reader 可以正确读取。但分隔符也许还会引起其他的什么问题。最后我们发现一个简单的 Helloworld 也需要写许多处理消息格式的代码。   
如果使用 Protobuf，那么这些细节就可以不需要应用程序来考虑了。   
使用 Protobuf，Writer 的工作很简单，需要处理的结构化数据由 .proto 文件描述，经过上一节中的编译过程后，该数据化结构对应了一个 C++ 的类，并定义在 lm.helloworld.pb.h 中。对于本例，类名为 lm::helloworld。   
Writer 需要 include 该头文件，然后便可以使用这个类了。   
现在，在 Writer 代码中，将要存入磁盘的结构化数据由一个 lm::helloworld 类的对象表示，它提供了一系列的 get/set 函数用来修改和读取结构化数据中的数据成员，或者叫 field。   
当我们需要将该结构化数据保存到磁盘上时，类 lm::helloworld 已经提供相应的方法来把一个复杂的数据变成一个字节序列，我们可以将这个字节序列写入磁盘。   
对于想要读取这个数据的程序来说，也只需要使用类 lm::helloworld 的相应反序列化方法来将这个字节序列重新转换会结构化数据。这同我们开始时那个“123”的想法类似，不过 Protobuf 想的远远比我们那个粗糙的字符串转换要全面，因此，我们不如放心将这类事情交给 Protobuf 吧。   
程序清单 2 演示了 Writer 的主要代码，您一定会觉得很简单吧？   
清单 2. Writer 的主要代码

#include "lm.helloworld.pb.h"

…

int main(void)

{

lm::helloworld msg1;

msg1.set\_id(101);

msg1.set\_str(“hello”);

// Write the new address book back to disk.

fstream output("./log", ios::out | ios::trunc | ios::binary);

if (!msg1.SerializeToOstream(&output)) {

cerr << "Failed to write msg." << endl;

return -1;

}

return 0;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

Msg1 是一个 helloworld 类的对象，set\_id() 用来设置 id 的值。SerializeToOstream 将对象序列化后写入一个 fstream 流。   
代码清单 3 列出了 reader 的主要代码。   
清单 3. Reader

#include "lm.helloworld.pb.h"

…

void ListMsg(const lm::helloworld & msg) {

cout << msg.id() << endl;

cout << msg.str() << endl;

}

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

lm::helloworld msg1;

{

fstream input("./log", ios::in | ios::binary);

if (!msg1.ParseFromIstream(&input)) {

cerr << "Failed to parse address book." << endl;

return -1;

}

}

ListMsg(msg1);

…

}

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

同样，Reader 声明类 helloworld 的对象 msg1，然后利用 ParseFromIstream 从一个 fstream 流中读取信息并反序列化。此后，ListMsg 中采用 get 方法读取消息的内部信息，并进行打印输出操作。   
**运行结果**   
运行 Writer 和 Reader 的结果如下：

\>writer

\>reader

101

Hello

* 1
* 2
* 3
* 4

Reader 读取文件 log 中的序列化信息并打印到屏幕上。本文中所有的例子代码都可以在附件中下载。您可以亲身体验一下。   
这个例子本身并无意义，但只要您稍加修改就可以将它变成更加有用的程序。比如将磁盘替换为网络 socket，那么就可以实现基于网络的数据交换任务。而存储和交换正是 Protobuf 最有效的应用领域。

### ****二、caffe.proto中的几个重要数据类型****

看完了上面关于protocol buffer的介绍，大家应该可以知道其实caffe.pb.cc里面的东西都是从caffe.proto编译而来的，无非就是一些关于这些数据结构（类）的标准化操作，比如

void CopyFrom();

void MergeFrom();

void CopyFrom();

void MergeFrom;

void Clear();

bool IsInitialized() const;

int ByteSize() const;

bool MergePartialFromCodedStream();

void SerializeWithCachedSizes() const;

SerializeWithCachedSizesToArray() const;

int GetCachedSize()

void SharedCtor();

void SharedDtor();

void SetCachedSize() const;

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

**<0> BlobProto**

message BlobProto {//blob的属性以及blob中的数据(data\diff)

optional int32 num = 1 [default = 0];

optional int32 channels = 2 [default = 0];

optional int32 height = 3 [default = 0];

optional int32 width = 4 [default = 0];

repeated float data = 5 [packed = true];

repeated float diff = 6 [packed = true];

}

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

**<1> Datum**

message Datum {

optional int32 channels = 1;

optional int32 height = 2;

optional int32 width = 3;

optional bytes data = 4;//真实的图像数据，以字节存储(bytes)

optional int32 label = 5;

repeated float float\_data = 6;//datum也能存float类型的数据(float)

}

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

**<2> LayerParameter**

message LayerParameter {

repeated string bottom = 2; //输入的blob的名字(string)

repeated string top = 3; //输出的blob的名字(string)

optional string name = 4; //层的名字

enum LayerType { //层的枚举（enum，和c++中的enum一样）

NONE = 0;

ACCURACY = 1;

BNLL = 2;

CONCAT = 3;

CONVOLUTION = 4;

DATA = 5;

DROPOUT = 6;

EUCLIDEAN\_LOSS = 7;

ELTWISE\_PRODUCT = 25;

FLATTEN = 8;

HDF5\_DATA = 9;

HDF5\_OUTPUT = 10;

HINGE\_LOSS = 28;

IM2COL = 11;

IMAGE\_DATA = 12;

INFOGAIN\_LOSS = 13;

INNER\_PRODUCT = 14;

LRN = 15;

MEMORY\_DATA = 29;

MULTINOMIAL\_LOGISTIC\_LOSS = 16;

POOLING = 17;

POWER = 26;

RELU = 18;

SIGMOID = 19;

SIGMOID\_CROSS\_ENTROPY\_LOSS = 27;

SOFTMAX = 20;

SOFTMAX\_LOSS = 21;

SPLIT = 22;

TANH = 23;

WINDOW\_DATA = 24;

}

optional LayerType type = 5; // 层的类型

repeated BlobProto blobs = 6; //blobs的数值参数

repeated float blobs\_lr = 7; //学习速率(repeated)，如果你想那个设置一个blob的学习速率，你需要设置所有blob的学习速率。

repeated float weight\_decay = 8; //权值衰减(repeated)

// 相对于某一特定层的参数(optional)

optional ConcatParameter concat\_param = 9;

optional ConvolutionParameter convolution\_param = 10;

optional DataParameter data\_param = 11;

optional DropoutParameter dropout\_param = 12;

optional HDF5DataParameter hdf5\_data\_param = 13;

optional HDF5OutputParameter hdf5\_output\_param = 14;

optional ImageDataParameter image\_data\_param = 15;

optional InfogainLossParameter infogain\_loss\_param = 16;

optional InnerProductParameter inner\_product\_param = 17;

optional LRNParameter lrn\_param = 18;

optional MemoryDataParameter memory\_data\_param = 22;

optional PoolingParameter pooling\_param = 19;

optional PowerParameter power\_param = 21;

optional WindowDataParameter window\_data\_param = 20;

optional V0LayerParameter layer = 1;

}

* 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

**<3> NetParameter**

message NetParameter {

optional string name = 1;//网络的名字

repeated LayerParameter layers = 2; //repeated类似于数组

repeated string input = 3;//输入层blob的名字

repeated int32 input\_dim = 4;//输入层blob的维度，应该等于(4\*#input)

optional bool force\_backward = 5 [default = false];//网络是否进行反向传播。如果设置为否，则由网络的结构和学习速率来决定是否进行反向传播。

}

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

**<4> SolverParameter**

message SolverParameter {

optional string train\_net = 1; // 训练网络的proto file

optional string test\_net = 2; // 测试网络的proto file

optional int32 test\_iter = 3 [default = 0]; // 每次测试时的迭代次数

optional int32 test\_interval = 4 [default = 0]; // 两次测试的间隔迭代次数

optional bool test\_compute\_loss = 19 [default = false];

optional float base\_lr = 5; // 基本学习率

optional int32 display = 6; // 两次显示的间隔迭代次数

optional int32 max\_iter = 7; // 最大迭代次数

optional string lr\_policy = 8; // 学习速率衰减方式

optional float gamma = 9; // 关于梯度下降的一个参数

optional float power = 10; // 计算学习率的一个参数

optional float momentum = 11; // 动量

optional float weight\_decay = 12; // 权值衰减

optional int32 stepsize = 13; // 学习速率的衰减步长

optional int32 snapshot = 14 [default = 0]; // snapshot的间隔

optional string snapshot\_prefix = 15; // snapshot的前缀

optional bool snapshot\_diff = 16 [default = false]; // 是否对于 diff 进行 snapshot

enum SolverMode {

CPU = 0;

GPU = 1;

}

optional SolverMode solver\_mode = 17 [default = GPU]; // solver的模式，默认为GPU

optional int32 device\_id = 18 [default = 0]; // GPU的ID

optional int64 random\_seed = 20 [default = -1]; // 随机数种子

}

* 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

### ****三、caffe.proto源码****

// Copyright 2014 BVLC and contributors.

package caffe;

message BlobProto {

optional int32 num = 1 [default = 0];

optional int32 channels = 2 [default = 0];

optional int32 height = 3 [default = 0];

optional int32 width = 4 [default = 0];

repeated float data = 5 [packed = true];

repeated float diff = 6 [packed = true];

}

// The BlobProtoVector is simply a way to pass multiple blobproto instances

// around.

message BlobProtoVector {

repeated BlobProto blobs = 1;

}

message Datum {

optional int32 channels = 1;

optional int32 height = 2;

optional int32 width = 3;

// the actual image data, in bytes

optional bytes data = 4;

optional int32 label = 5;

// Optionally, the datum could also hold float data.

repeated float float\_data = 6;

}

message FillerParameter {

// The filler type.

optional string type = 1 [default = 'constant'];

optional float value = 2 [default = 0]; // the value in constant filler

optional float min = 3 [default = 0]; // the min value in uniform filler

optional float max = 4 [default = 1]; // the max value in uniform filler

optional float mean = 5 [default = 0]; // the mean value in Gaussian filler

optional float std = 6 [default = 1]; // the std value in Gaussian filler

// The expected number of non-zero input weights for a given output in

// Gaussian filler -- the default -1 means don't perform sparsification.

optional int32 sparse = 7 [default = -1];

}

message NetParameter {

optional string name = 1; // consider giving the network a name

repeated LayerParameter layers = 2; // a bunch of layers.

// The input blobs to the network.

repeated string input = 3;

// The dim of the input blobs. For each input blob there should be four

// values specifying the num, channels, height and width of the input blob.

// Thus, there should be a total of (4 \* #input) numbers.

repeated int32 input\_dim = 4;

// Whether the network will force every layer to carry out backward operation.

// If set False, then whether to carry out backward is determined

// automatically according to the net structure and learning rates.

optional bool force\_backward = 5 [default = false];

}

message SolverParameter {

optional string train\_net = 1; // The proto file for the training net.

optional string test\_net = 2; // The proto file for the testing net.

// The number of iterations for each testing phase.

optional int32 test\_iter = 3 [default = 0];

// The number of iterations between two testing phases.

optional int32 test\_interval = 4 [default = 0];

optional bool test\_compute\_loss = 19 [default = false];

optional float base\_lr = 5; // The base learning rate

// the number of iterations between displaying info. If display = 0, no info

// will be displayed.

optional int32 display = 6;

optional int32 max\_iter = 7; // the maximum number of iterations

optional string lr\_policy = 8; // The learning rate decay policy.

optional float gamma = 9; // The parameter to compute the learning rate.

optional float power = 10; // The parameter to compute the learning rate.

optional float momentum = 11; // The momentum value.

optional float weight\_decay = 12; // The weight decay.

optional int32 stepsize = 13; // the stepsize for learning rate policy "step"

optional int32 snapshot = 14 [default = 0]; // The snapshot interval

optional string snapshot\_prefix = 15; // The prefix for the snapshot.

// whether to snapshot diff in the results or not. Snapshotting diff will help

// debugging but the final protocol buffer size will be much larger.

optional bool snapshot\_diff = 16 [default = false];

// the mode solver will use: 0 for CPU and 1 for GPU. Use GPU in default.

enum SolverMode {

CPU = 0;

GPU = 1;

}

optional SolverMode solver\_mode = 17 [default = GPU];

// the device\_id will that be used in GPU mode. Use device\_id = 0 in default.

optional int32 device\_id = 18 [default = 0];

// If non-negative, the seed with which the Solver will initialize the Caffe

// random number generator -- useful for reproducible results. Otherwise,

// (and by default) initialize using a seed derived from the system clock.

optional int64 random\_seed = 20 [default = -1];

}

// A message that stores the solver snapshots

message SolverState {

optional int32 iter = 1; // The current iteration

optional string learned\_net = 2; // The file that stores the learned net.

repeated BlobProto history = 3; // The history for sgd solvers

}

// Update the next available ID when you add a new LayerParameter field.

//

// LayerParameter next available ID: 23 (last added: memory\_data\_param)

message LayerParameter {

repeated string bottom = 2; // the name of the bottom blobs

repeated string top = 3; // the name of the top blobs

optional string name = 4; // the layer name

// Add new LayerTypes to the enum below in lexicographical order (other than

// starting with NONE), starting with the next available ID in the comment

// line above the enum. Update the next available ID when you add a new

// LayerType.

//

// LayerType next available ID: 30 (last added: MEMORY\_DATA)

enum LayerType {

// "NONE" layer type is 0th enum element so that we don't cause confusion

// by defaulting to an existent LayerType (instead, should usually error if

// the type is unspecified).

NONE = 0;

ACCURACY = 1;

BNLL = 2;

CONCAT = 3;

CONVOLUTION = 4;

DATA = 5;

DROPOUT = 6;

EUCLIDEAN\_LOSS = 7;

ELTWISE\_PRODUCT = 25;

FLATTEN = 8;

HDF5\_DATA = 9;

HDF5\_OUTPUT = 10;

HINGE\_LOSS = 28;

IM2COL = 11;

IMAGE\_DATA = 12;

INFOGAIN\_LOSS = 13;

INNER\_PRODUCT = 14;

LRN = 15;

MEMORY\_DATA = 29;

MULTINOMIAL\_LOGISTIC\_LOSS = 16;

POOLING = 17;

POWER = 26;

RELU = 18;

SIGMOID = 19;

SIGMOID\_CROSS\_ENTROPY\_LOSS = 27;

SOFTMAX = 20;

SOFTMAX\_LOSS = 21;

SPLIT = 22;

TANH = 23;

WINDOW\_DATA = 24;

}

optional LayerType type = 5; // the layer type from the enum above

// The blobs containing the numeric parameters of the layer

repeated BlobProto blobs = 6;

// The ratio that is multiplied on the global learning rate. If you want to

// set the learning ratio for one blob, you need to set it for all blobs.

repeated float blobs\_lr = 7;

// The weight decay that is multiplied on the global weight decay.

repeated float weight\_decay = 8;

// Parameters for particular layer types.

optional ConcatParameter concat\_param = 9;

optional ConvolutionParameter convolution\_param = 10;

optional DataParameter data\_param = 11;

optional DropoutParameter dropout\_param = 12;

optional HDF5DataParameter hdf5\_data\_param = 13;

optional HDF5OutputParameter hdf5\_output\_param = 14;

optional ImageDataParameter image\_data\_param = 15;

optional InfogainLossParameter infogain\_loss\_param = 16;

optional InnerProductParameter inner\_product\_param = 17;

optional LRNParameter lrn\_param = 18;

optional MemoryDataParameter memory\_data\_param = 22;

optional PoolingParameter pooling\_param = 19;

optional PowerParameter power\_param = 21;

optional WindowDataParameter window\_data\_param = 20;

// DEPRECATED: The layer parameters specified as a V0LayerParameter.

// This should never be used by any code except to upgrade to the new

// LayerParameter specification.

optional V0LayerParameter layer = 1;

}

// Message that stores parameters used by ConcatLayer

message ConcatParameter {

// Concat Layer needs to specify the dimension along the concat will happen,

// the other dimensions must be the same for all the bottom blobs

// By default it will concatenate blobs along channels dimension

optional uint32 concat\_dim = 1 [default = 1];

}

// Message that stores parameters used by ConvolutionLayer

message ConvolutionParameter {

optional uint32 num\_output = 1; // The number of outputs for the layer

optional bool bias\_term = 2 [default = true]; // whether to have bias terms

optional uint32 pad = 3 [default = 0]; // The padding size

optional uint32 kernel\_size = 4; // The kernel size

optional uint32 group = 5 [default = 1]; // The group size for group conv

optional uint32 stride = 6 [default = 1]; // The stride

optional FillerParameter weight\_filler = 7; // The filler for the weight

optional FillerParameter bias\_filler = 8; // The filler for the bias

}

// Message that stores parameters used by DataLayer

message DataParameter {

// Specify the data source.

optional string source = 1;

// For data pre-processing, we can do simple scaling and subtracting the

// data mean, if provided. Note that the mean subtraction is always carried

// out before scaling.

optional float scale = 2 [default = 1];

optional string mean\_file = 3;

// Specify the batch size.

optional uint32 batch\_size = 4;

// Specify if we would like to randomly crop an image.

optional uint32 crop\_size = 5 [default = 0];

// Specify if we want to randomly mirror data.

optional bool mirror = 6 [default = false];

// The rand\_skip variable is for the data layer to skip a few data points

// to avoid all asynchronous sgd clients to start at the same point. The skip

// point would be set as rand\_skip \* rand(0,1). Note that rand\_skip should not

// be larger than the number of keys in the leveldb.

optional uint32 rand\_skip = 7 [default = 0];

}

// Message that stores parameters used by DropoutLayer

message DropoutParameter {

optional float dropout\_ratio = 1 [default = 0.5]; // dropout ratio

}

// Message that stores parameters used by HDF5DataLayer

message HDF5DataParameter {

// Specify the data source.

optional string source = 1;

// Specify the batch size.

optional uint32 batch\_size = 2;

}

// Message that stores parameters used by HDF5OutputLayer

message HDF5OutputParameter {

optional string file\_name = 1;

}

// Message that stores parameters used by ImageDataLayer

message ImageDataParameter {

// Specify the data source.

optional string source = 1;

// For data pre-processing, we can do simple scaling and subtracting the

// data mean, if provided. Note that the mean subtraction is always carried

// out before scaling.

optional float scale = 2 [default = 1];

optional string mean\_file = 3;

// Specify the batch size.

optional uint32 batch\_size = 4;

// Specify if we would like to randomly crop an image.

optional uint32 crop\_size = 5 [default = 0];

// Specify if we want to randomly mirror data.

optional bool mirror = 6 [default = false];

// The rand\_skip variable is for the data layer to skip a few data points

// to avoid all asynchronous sgd clients to start at the same point. The skip

// point would be set as rand\_skip \* rand(0,1). Note that rand\_skip should not

// be larger than the number of keys in the leveldb.

optional uint32 rand\_skip = 7 [default = 0];

// Whether or not ImageLayer should shuffle the list of files at every epoch.

optional bool shuffle = 8 [default = false];

// It will also resize images if new\_height or new\_width are not zero.

optional uint32 new\_height = 9 [default = 0];

optional uint32 new\_width = 10 [default = 0];

}

// Message that stores parameters InfogainLossLayer

message InfogainLossParameter {

// Specify the infogain matrix source.

optional string source = 1;

}

// Message that stores parameters used by InnerProductLayer

message InnerProductParameter {

optional uint32 num\_output = 1; // The number of outputs for the layer

optional bool bias\_term = 2 [default = true]; // whether to have bias terms

optional FillerParameter weight\_filler = 3; // The filler for the weight

optional FillerParameter bias\_filler = 4; // The filler for the bias

}

// Message that stores parameters used by LRNLayer

message LRNParameter {

optional uint32 local\_size = 1 [default = 5];

optional float alpha = 2 [default = 1.];

optional float beta = 3 [default = 0.75];

enum NormRegion {

ACROSS\_CHANNELS = 0;

WITHIN\_CHANNEL = 1;

}

optional NormRegion norm\_region = 4 [default = ACROSS\_CHANNELS];

}

// Message that stores parameters used by MemoryDataLayer

message MemoryDataParameter {

optional uint32 batch\_size = 1;

optional uint32 channels = 2;

optional uint32 height = 3;

optional uint32 width = 4;

}

// Message that stores parameters used by PoolingLayer

message PoolingParameter {

enum PoolMethod {

MAX = 0;

AVE = 1;

STOCHASTIC = 2;

}

optional PoolMethod pool = 1 [default = MAX]; // The pooling method

optional uint32 kernel\_size = 2; // The kernel size

optional uint32 stride = 3 [default = 1]; // The stride

// The padding size -- currently implemented only for average pooling.

optional uint32 pad = 4 [default = 0];

}

// Message that stores parameters used by PowerLayer

message PowerParameter {

// PowerLayer computes outputs y = (shift + scale \* x) ^ power.

optional float power = 1 [default = 1.0];

optional float scale = 2 [default = 1.0];

optional float shift = 3 [default = 0.0];

}

// Message that stores parameters used by WindowDataLayer

message WindowDataParameter {

// Specify the data source.

optional string source = 1;

// For data pre-processing, we can do simple scaling and subtracting the

// data mean, if provided. Note that the mean subtraction is always carried

// out before scaling.

optional float scale = 2 [default = 1];

optional string mean\_file = 3;

// Specify the batch size.

optional uint32 batch\_size = 4;

// Specify if we would like to randomly crop an image.

optional uint32 crop\_size = 5 [default = 0];

// Specify if we want to randomly mirror data.

optional bool mirror = 6 [default = false];

// Foreground (object) overlap threshold

optional float fg\_threshold = 7 [default = 0.5];

// Background (non-object) overlap threshold

optional float bg\_threshold = 8 [default = 0.5];

// Fraction of batch that should be foreground objects

optional float fg\_fraction = 9 [default = 0.25];

// Amount of contextual padding to add around a window

// (used only by the window\_data\_layer)

optional uint32 context\_pad = 10 [default = 0];

// Mode for cropping out a detection window

// warp: cropped window is warped to a fixed size and aspect ratio

// square: the tightest square around the window is cropped

optional string crop\_mode = 11 [default = "warp"];

}

// DEPRECATED: V0LayerParameter is the old way of specifying layer parameters

// in Caffe. We keep this message type around for legacy support.

message V0LayerParameter {

optional string name = 1; // the layer name

optional string type = 2; // the string to specify the layer type

// Parameters to specify layers with inner products.

optional uint32 num\_output = 3; // The number of outputs for the layer

optional bool biasterm = 4 [default = true]; // whether to have bias terms

optional FillerParameter weight\_filler = 5; // The filler for the weight

optional FillerParameter bias\_filler = 6; // The filler for the bias

optional uint32 pad = 7 [default = 0]; // The padding size

optional uint32 kernelsize = 8; // The kernel size

optional uint32 group = 9 [default = 1]; // The group size for group conv

optional uint32 stride = 10 [default = 1]; // The stride

enum PoolMethod {

MAX = 0;

AVE = 1;

STOCHASTIC = 2;

}

optional PoolMethod pool = 11 [default = MAX]; // The pooling method

optional float dropout\_ratio = 12 [default = 0.5]; // dropout ratio

optional uint32 local\_size = 13 [default = 5]; // for local response norm

optional float alpha = 14 [default = 1.]; // for local response norm

optional float beta = 15 [default = 0.75]; // for local response norm

// For data layers, specify the data source

optional string source = 16;

// For data pre-processing, we can do simple scaling and subtracting the

// data mean, if provided. Note that the mean subtraction is always carried

// out before scaling.

optional float scale = 17 [default = 1];

optional string meanfile = 18;

// For data layers, specify the batch size.

optional uint32 batchsize = 19;

// For data layers, specify if we would like to randomly crop an image.

optional uint32 cropsize = 20 [default = 0];

// For data layers, specify if we want to randomly mirror data.

optional bool mirror = 21 [default = false];

// The blobs containing the numeric parameters of the layer

repeated BlobProto blobs = 50;

// The ratio that is multiplied on the global learning rate. If you want to

// set the learning ratio for one blob, you need to set it for all blobs.

repeated float blobs\_lr = 51;

// The weight decay that is multiplied on the global weight decay.

repeated float weight\_decay = 52;

// The rand\_skip variable is for the data layer to skip a few data points

// to avoid all asynchronous sgd clients to start at the same point. The skip

// point would be set as rand\_skip \* rand(0,1). Note that rand\_skip should not

// be larger than the number of keys in the leveldb.

optional uint32 rand\_skip = 53 [default = 0];

// Fields related to detection (det\_\*)

// foreground (object) overlap threshold

optional float det\_fg\_threshold = 54 [default = 0.5];

// background (non-object) overlap threshold

optional float det\_bg\_threshold = 55 [default = 0.5];

// Fraction of batch that should be foreground objects

optional float det\_fg\_fraction = 56 [default = 0.25];

// optional bool OBSOLETE\_can\_clobber = 57 [default = true];

// Amount of contextual padding to add around a window

// (used only by the window\_data\_layer)

optional uint32 det\_context\_pad = 58 [default = 0];

// Mode for cropping out a detection window

// warp: cropped window is warped to a fixed size and aspect ratio

// square: the tightest square around the window is cropped

optional string det\_crop\_mode = 59 [default = "warp"];

// For ReshapeLayer, one needs to specify the new dimensions.

optional int32 new\_num = 60 [default = 0];

optional int32 new\_channels = 61 [default = 0];

optional int32 new\_height = 62 [default = 0];

optional int32 new\_width = 63 [default = 0];

// Whether or not ImageLayer should shuffle the list of files at every epoch.

// It will also resize images if new\_height or new\_width are not zero.

optional bool shuffle\_images = 64 [default = false];

// For ConcatLayer, one needs to specify the dimension for concatenation, and

// the other dimensions must be the same for all the bottom blobs.

// By default it will concatenate blobs along the channels dimension.

optional uint32 concat\_dim = 65 [default = 1];

optional HDF5OutputParameter hdf5\_output\_param = 1001;

}

# [caffe&windows工作过程 — cifar10为例](http://blog.csdn.net/qq_16055159/article/details/45130509)

本章主要介绍caffe的整个工程执行过程，错误之处望大家 **指出** 并 **海涵**。

* 标红的表示关键代码
* 不同的颜色表示下面各函数所属的类
* 箭头指向对应的函数实现
* 本文主要涉及3个.cpp文件：train\_net.cpp、solver.cpp、net.cpp

## caffe工程流程图



# [反向传导算法 & SOFTMAX & Innerproduct\_layer.cpp & softmax\_loss\_layer.cpp](http://blog.csdn.net/qq_16055159/article/details/45952915)

好不容易从云笔记搞到了CSDN上…   
请放大至125%观看，效果更佳。   
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
