Caffe-代码解析-Blob

时间：2015-06-04 22:46:30 阅读：93 评论：0 收藏：0 | 收藏+]

标签：des class com 代码 使用 si java it la

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
```

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();
```

↑

↓

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 {
```

分享档案 更多>

2015年08月31日 (1486)
2015年08月30日 (2038)
2015年08月29日 (1707)
2015年08月28日 (2534)
2015年08月27日 (2580)
2015年08月26日 (2441)
2015年08月25日 (2432)
2015年08月24日 (4)
2015年08月21日 (2194)
2015年08月20日 (2359)

文章周排行 更多>

1. "The identity used to sign the executable is no longer valid"错误解决方法 2015-07-20
2. 利用Wireshark截取数据包，并对数据包进行解析 2014-05-10
3. HC-05 蓝牙模块的调试与使用 2015-03-04
4. Bootstrap-用ICheck插件给CheckBox换新装 2015-03-05
5. 登录失败, 密码错误或者IMAP服务未开通 2015-03-07
6. 初次体验VS2015正式版，安装详细过程。 2015-07-22
7. KeepAlive详解 2014-07-17
8. Xcode插件之Alcatraz的安装和遇到的问题 2015-05-08
9. 寒龙国内网所有高级代理-全部高速代理IP。欢迎使用！ 2014-12-09
10. zepto的tap事件的点透问题的几种解决方案 2015-03-02

最新新闻 更多

1. 《巫师3》制作人表示昆特牌系统或将有好消息 2015-08-31
2. 武汉特斯拉车主“伤心体验”之痛 2015-08-31
3. 首个中间件工程类大赛在阿里举办 浙大冠军获10万大奖 2015-08-31
4. [视频]苹果在VMA期间发布两段Apple Music广告 2015-08-31
5. 人人成《碟中谍5》国内独家合作社交网站 2015-08-31

```
  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。
```

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-Blob

标签 : des class com 代码 使用 si java it la