kaldi-vector.h

Go to the documentation of this file.

```
// matrix/kaldi-vector.h

// Copyright 2009-2012   Ondrej Glembek;  Microsoft Corporation;  Lukas Burget;
//                       Saarland University (Author: Arnab Ghoshal);
//                       Ariya Rastrow;  Petr Schwarz;  Yanmin Qian;
//                       Karel Vesely;  Go Vivace Inc.;  Arnab Ghoshal
//                       Wei Shi;
//                2015   Guoguo Chen

// See ../../COPYING for clarification regarding multiple authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
// WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
// MERCHANTABLITY OR NON-INFRINGEMENT.
// See the Apache 2 License for the specific language governing permissions and
// limitations under the License.

#ifndef KALDI_MATRIX_KALDI_VECTOR_H_
#define KALDI_MATRIX_KALDI_VECTOR_H_ 1

#include "matrix/matrix-common.h"

namespace kaldi {


template<typename Real>
class VectorBase {
 public:
  void SetZero();

  bool IsZero(Real cutoff = 1.0e-06) const;     // replace magic number

  void Set(Real f);

  void SetRandn();

  MatrixIndexT RandCategorical() const;

  inline MatrixIndexT Dim() const { return dim_; }

  inline MatrixIndexT SizeInBytes() const { return (dim_*sizeof(Real)); }

  inline Real* Data() { return data_; }

  inline const Real* Data() const { return data_; }

  inline Real operator() (MatrixIndexT i) const {
    KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
                 static_cast<UnsignedMatrixIndexT>(dim_));
    return *(data_ + i);
  }

  inline Real & operator() (MatrixIndexT i) {
    KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
                 static_cast<UnsignedMatrixIndexT>(dim_));
    return *(data_ + i);
  }

  SubVector<Real> Range(const MatrixIndexT o, const MatrixIndexT l) {
    return SubVector<Real>(*this, o, l);
  }
```

```cpp

  const SubVector<Real> Range(const MatrixIndexT o,
                              const MatrixIndexT l) const {
    return SubVector<Real>(*this, o, l);
  }

  void CopyFromVec(const VectorBase<Real> &v);

  template<typename OtherReal>
  void CopyFromPacked(const PackedMatrix<OtherReal> &M);

  template<typename OtherReal>
  void CopyFromVec(const VectorBase<OtherReal> &v);

  template<typename OtherReal>
  void CopyFromVec(const CuVectorBase<OtherReal> &v);


  void ApplyLog();

  void ApplyLogAndCopy(const VectorBase<Real> &v);

  void ApplyExp();

  void ApplyAbs();

  MatrixIndexT ApplyFloor(Real floor_val);

  MatrixIndexT ApplyCeiling(Real ceil_val);

  MatrixIndexT ApplyFloor(const VectorBase<Real> &floor_vec);

  Real ApplySoftMax();

  Real ApplyLogSoftMax();

  void Tanh(const VectorBase<Real> &src);

  void Sigmoid(const VectorBase<Real> &src);

  void ApplyPow(Real power);

  void ApplyPowAbs(Real power, bool include_sign=false);

  Real Norm(Real p) const;

  bool ApproxEqual(const VectorBase<Real> &other, float tol = 0.01) const;

  void InvertElements();

  template<typename OtherReal>
  void AddVec(const Real alpha, const VectorBase<OtherReal> &v);

  void AddVec2(const Real alpha, const VectorBase<Real> &v);

  template<typename OtherReal>
  void AddVec2(const Real alpha, const VectorBase<OtherReal> &v);

  void AddMatVec(const Real alpha, const MatrixBase<Real> &M,
                 const MatrixTransposeType trans,  const VectorBase<Real> &v,
                 const Real beta); // **beta previously defaulted to 0.0**

  void AddMatSvec(const Real alpha, const MatrixBase<Real> &M,
                  const MatrixTransposeType trans,  const VectorBase<Real> &v,
                  const Real beta); // **beta previously defaulted to 0.0**


  void AddSpVec(const Real alpha, const SpMatrix<Real> &M,
                const VectorBase<Real> &v, const Real beta);  // **beta previously
defaulted to 0.0**

  void AddTpVec(const Real alpha, const TpMatrix<Real> &M,
                const MatrixTransposeType trans, const VectorBase<Real> &v,
                const Real beta);  // **beta previously defaulted to 0.0**

  void ReplaceValue(Real orig, Real changed);

  void MulElements(const VectorBase<Real> &v);
  template<typename OtherReal>
  void MulElements(const VectorBase<OtherReal> &v);

  void DivElements(const VectorBase<Real> &v);
  template<typename OtherReal>
  void DivElements(const VectorBase<OtherReal> &v);

  void Add(Real c);

  //  this <-- alpha * v .* r + beta*this .
  void AddVecVec(Real alpha, const VectorBase<Real> &v,
                 const VectorBase<Real> &r, Real beta);

  void AddVecDivVec(Real alpha, const VectorBase<Real> &v,
                    const VectorBase<Real> &r, Real beta);

  void Scale(Real alpha);

  void MulTp(const TpMatrix<Real> &M, const MatrixTransposeType trans);

  void Solve(const TpMatrix<Real> &M, const MatrixTransposeType trans);

  void CopyRowsFromMat(const MatrixBase<Real> &M);
  template<typename OtherReal>
  void CopyRowsFromMat(const MatrixBase<OtherReal> &M);

  void CopyRowsFromMat(const CuMatrixBase<Real> &M);

  void CopyColsFromMat(const MatrixBase<Real> &M);

  void CopyRowFromMat(const MatrixBase<Real> &M, MatrixIndexT row);
  template<typename OtherReal>
  void CopyRowFromMat(const MatrixBase<OtherReal> &M, MatrixIndexT row);

  template<typename OtherReal>
  void CopyRowFromSp(const SpMatrix<OtherReal> &S, MatrixIndexT row);

  template<typename OtherReal>
  void CopyColFromMat(const MatrixBase<OtherReal> &M , MatrixIndexT col);

  void CopyDiagFromMat(const MatrixBase<Real> &M);

  void CopyDiagFromPacked(const PackedMatrix<Real> &M);


  inline void CopyDiagFromSp(const SpMatrix<Real> &M) { CopyDiagFromPacked(M); }

  inline void CopyDiagFromTp(const TpMatrix<Real> &M) { CopyDiagFromPacked(M); }

  Real Max() const;

  Real Max(MatrixIndexT *index) const;

  Real Min() const;

  Real Min(MatrixIndexT *index) const;

  Real Sum() const;

  Real SumLog() const;

  void AddRowSumMat(Real alpha, const MatrixBase<Real> &M, Real beta = 1.0);

  void AddColSumMat(Real alpha, const MatrixBase<Real> &M, Real beta = 1.0);

  void AddDiagMat2(Real alpha, const MatrixBase<Real> &M,
                   MatrixTransposeType trans = kNoTrans, Real beta = 1.0);

  void AddDiagMatMat(Real alpha, const MatrixBase<Real> &M, MatrixTransposeType
transM,
                     const MatrixBase<Real> &N, MatrixTransposeType transN,
                     Real beta = 1.0);

  Real LogSumExp(Real prune = -1.0) const;

  void Read(std::istream & in, bool binary, bool add = false);

  void Write(std::ostream &Out, bool binary) const;

  friend class VectorBase<double>;
  friend class VectorBase<float>;
  friend class CuVectorBase<Real>;
  friend class CuVector<Real>;
 protected:
  ~VectorBase() {}

  explicit VectorBase(): data_(NULL), dim_(0) {
    KALDI_ASSERT_IS_FLOATING_TYPE(Real);
  }

// Took this out since it is not currently used, and it is possible to create
// objects where the allocated memory is not the same size as dim_ : Arnab
//  /// Initializer from a pointer and a size; keeps the pointer internally
//  /// (ownership or non-ownership depends on the child class).
//  explicit VectorBase(Real* data, MatrixIndexT dim)
//      : data_(data), dim_(dim) {}

  // Arnab : made this protected since it is unsafe too.
  void CopyFromPtr(const Real* Data, MatrixIndexT sz);

  Real* data_;
  MatrixIndexT dim_;
  KALDI_DISALLOW_COPY_AND_ASSIGN(VectorBase);
}; // class VectorBase

template<typename Real>
class Vector: public VectorBase<Real> {
 public:
  Vector(): VectorBase<Real>() {}

  explicit Vector(const MatrixIndexT s,
                  MatrixResizeType resize_type = kSetZero)
      : VectorBase<Real>() {  Resize(s, resize_type);  }

  template<typename OtherReal>
  explicit Vector(const CuVectorBase<OtherReal> &cu);

  Vector(const Vector<Real> &v) : VectorBase<Real>()  { //  (cannot be explicit)
    Resize(v.Dim(), kUndefined);
    this->CopyFromVec(v);
  }

  explicit Vector(const VectorBase<Real> &v) : VectorBase<Real>() {
    Resize(v.Dim(), kUndefined);
    this->CopyFromVec(v);
  }

  template<typename OtherReal>
  explicit Vector(const VectorBase<OtherReal> &v): VectorBase<Real>() {
    Resize(v.Dim(), kUndefined);
    this->CopyFromVec(v);
  }

// Took this out since it is unsafe : Arnab
//  /// Constructor from a pointer and a size; copies the data to a location
//  /// it owns.
//  Vector(const Real* Data, const MatrixIndexT s): VectorBase<Real>() {
//    Resize(s);
  //    CopyFromPtr(Data, s);
//  }


  void Swap(Vector<Real> *other);

  ~Vector() { Destroy(); }

  void Read(std::istream & in, bool binary, bool add = false);

  void Resize(MatrixIndexT length, MatrixResizeType resize_type = kSetZero);

  void RemoveElement(MatrixIndexT i);

  Vector<Real> &operator = (const Vector<Real> &other) {
    Resize(other.Dim(), kUndefined);
    this->CopyFromVec(other);
    return *this;
  }

  Vector<Real> &operator = (const VectorBase<Real> &other) {
    Resize(other.Dim(), kUndefined);
    this->CopyFromVec(other);
    return *this;
  }
 private:
  void Init(const MatrixIndexT dim);

  void Destroy();

};


template<typename Real>
class SubVector : public VectorBase<Real> {
 public:
  SubVector(const VectorBase<Real> &t, const MatrixIndexT origin,
            const MatrixIndexT length) : VectorBase<Real>() {
    // following assert equiv to origin>=0 && length>=0 &&
    // origin+length <= rt.dim_
    KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(origin)+
                 static_cast<UnsignedMatrixIndexT>(length) <=
                 static_cast<UnsignedMatrixIndexT>(t.Dim()));
    VectorBase<Real>::data_ = const_cast<Real*> (t.Data()+origin);
    VectorBase<Real>::dim_   = length;
  }

  SubVector(const PackedMatrix<Real> &M) {
    VectorBase<Real>::data_ = const_cast<Real*> (M.Data());
    VectorBase<Real>::dim_   = (M.NumRows()*(M.NumRows()+1))/2;
  }

  SubVector(const SubVector &other) : VectorBase<Real> () {
    // this copy constructor needed for Range() to work in base class.
    VectorBase<Real>::data_ = other.data_;
    VectorBase<Real>::dim_ = other.dim_;
  }

  SubVector(Real *data, MatrixIndexT length) : VectorBase<Real> () {
    VectorBase<Real>::data_ = data;
    VectorBase<Real>::dim_   = length;
  }


  SubVector(const MatrixBase<Real> &matrix, MatrixIndexT row) {
    VectorBase<Real>::data_ = const_cast<Real*>(matrix.RowData(row));
    VectorBase<Real>::dim_   = matrix.NumCols();
  }

  ~SubVector() {}

 private:
  SubVector & operator = (const SubVector &other) {}
};

template<typename Real>
std::ostream & operator << (std::ostream & out, const VectorBase<Real> & v);

template<typename Real>
std::istream & operator >> (std::istream & in, VectorBase<Real> & v);

template<typename Real>
std::istream & operator >> (std::istream & in, Vector<Real> & v);

```

```


template<typename Real>
bool ApproxEqual(const VectorBase<Real> &a,
                 const VectorBase<Real> &b, Real tol = 0.01) {
  return a.ApproxEqual(b, tol);
}

template<typename Real>
inline void AssertEqual(VectorBase<Real> &a, VectorBase<Real> &b,
                        float tol = 0.01) {
  KALDI_ASSERT(a.ApproxEqual(b, tol));
}


template<typename Real>
Real VecVec(const VectorBase<Real> &v1, const VectorBase<Real> &v2);

template<typename Real, typename OtherReal>
Real VecVec(const VectorBase<Real> &v1, const VectorBase<OtherReal> &v2);


template<typename Real>
Real VecMatVec(const VectorBase<Real> &v1, const MatrixBase<Real> &M,
               const VectorBase<Real> &v2);



}  // namespace kaldi

// we need to include the implementation
#include "matrix/kaldi-vector-inl.h"



#endif  // KALDI_MATRIX_KALDI_VECTOR_H_

```