

kaldi-vector.h

Go to the documentation of this file.

```
1 // matrix/kaldi-vector.h
2
3 // Copyright 2009-2012 Ondrej Glembek; Microsoft Corporation; Lukas Burget;
4 // Saarland University (Author: Arnab Ghoshal);
5 // Ariya Rastrow; Petr Schwarz; Yanmin Qian;
6 // Karel Vesely; Go Vivace Inc.; Arnab Ghoshal
7 // Wei Shi;
8 // 2015 Guoguo Chen
9
10 // See ../../COPYING for clarification regarding multiple authors
11 //
12 // Licensed under the Apache License, Version 2.0 (the "License");
13 // you may not use this file except in compliance with the License.
14 // You may obtain a copy of the License at
15 //
16 // http://www.apache.org/licenses/LICENSE-2.0
17 //
18 // THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
19 // KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
20 // WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
21 // MERCHANTABILITY OR NON-INFRINGEMENT.
22 // See the Apache 2 License for the specific language governing permissions and
23 // limitations under the License.
24
25 #ifndef KALDI_MATRIX_KALDI_VECTOR_H_
26 #define KALDI_MATRIX_KALDI_VECTOR_H_ 1
27
28 #include "matrix/matrix-common.h"
29
30 namespace kaldi {
31
32
33
34
35
36
37
38 template<typename Real>
39 class VectorBase {
40 public:
41   void SetZero();
42   void SetZero();
43
44   bool IsZero(Real cutoff = 1.0e-06) const; // replace magic number
45
46   void Set(Real f);
47
48   void SetRandn();
49
50   MatrixIndexT RandCategorical() const;
51
52   inline MatrixIndexT Dim() const { return dim_; }
53
54   inline MatrixIndexT SizeInBytes() const { return (dim_*sizeof(Real)); }
55
56   inline Real* Data() { return data_; }
57
58   inline const Real* Data() const { return data_; }
59
60   inline Real operator() (MatrixIndexT i) const {
61     KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
62                           static_cast<UnsignedMatrixIndexT>(dim_));
63     return *(data_ + i);
64   }
65
66   inline Real & operator() (MatrixIndexT i) {
67     KALDI_PARANOID_ASSERT(static_cast<UnsignedMatrixIndexT>(i) <
68                           static_cast<UnsignedMatrixIndexT>(dim_));
69     return *(data_ + i);
70   }
71
72   SubVector<Real> Range(const MatrixIndexT o, const MatrixIndexT l) {
73     return SubVector<Real>(*this, o, l);
74   }
75 }
```

```

92
93     const SubVector<Real> Range(const MatrixIndexT o,
94                                const MatrixIndexT l) const {
95         return SubVector<Real>(*this, o, l);
96     }
97
98     void CopyFromVec(const VectorBase<Real> &v);
99
100     template<typename OtherReal>
101     void CopyFromPacked(const PackedMatrix<OtherReal> &M);
102
103     template<typename OtherReal>
104     void CopyFromVec(const VectorBase<OtherReal> &v);
105
106     template<typename OtherReal>
107     void CopyFromVec(const CuVectorBase<OtherReal> &v);
108
109
110     void ApplyLog();
111
112     void ApplyLogAndCopy(const VectorBase<Real> &v);
113
114     void ApplyExp();
115
116     void ApplyAbs();
117
118     MatrixIndexT ApplyFloor(Real floor_val);
119
120     MatrixIndexT ApplyCeiling(Real ceil_val);
121
122     MatrixIndexT ApplyFloor(const VectorBase<Real> &floor_vec);
123
124     Real ApplySoftMax();
125
126     Real ApplyLogSoftMax();
127
128     void Tanh(const VectorBase<Real> &src);
129
130     void Sigmoid(const VectorBase<Real> &src);
131
132     void ApplyPow(Real power);
133
134     void ApplyPowAbs(Real power, bool include_sign=false);
135
136     Real Norm(Real p) const;
137
138     bool ApproxEqual(const VectorBase<Real> &other, float tol = 0.01) const;
139
140     void InvertElements();
141
142     template<typename OtherReal>
143     void AddVec(const Real alpha, const VectorBase<OtherReal> &v);
144
145     void AddVec2(const Real alpha, const VectorBase<Real> &v);
146
147     template<typename OtherReal>
148     void AddVec2(const Real alpha, const VectorBase<OtherReal> &v);
149
150     void AddMatVec(const Real alpha, const MatrixBase<Real> &M,
151                   const MatrixTransposeType trans, const VectorBase<Real> &v,
152                   const Real beta); // **beta previously defaulted to 0.0**
153
154     void AddMatSvec(const Real alpha, const MatrixBase<Real> &M,
155                    const MatrixTransposeType trans, const VectorBase<Real> &v,
156                    const Real beta); // **beta previously defaulted to 0.0**
157
158     void AddSpVec(const Real alpha, const SpMatrix<Real> &M,
159                  const VectorBase<Real> &v, const Real beta); // **beta previously
160 defaulted to 0.0**
161
162     void AddTpVec(const Real alpha, const TpMatrix<Real> &M,
163                  const MatrixTransposeType trans, const VectorBase<Real> &v,
164                  const Real beta); // **beta previously defaulted to 0.0**
165
166

```

```

213 void ReplaceValue(Real orig, Real changed);
214
216 void MulElements(const VectorBase<Real> &v);
218 template<typename OtherReal>
219 void MulElements(const VectorBase<OtherReal> &v);
220
222 void DivElements(const VectorBase<Real> &v);
224 template<typename OtherReal>
225 void DivElements(const VectorBase<OtherReal> &v);
226
228 void Add(Real c);
229
231 // this <-- alpha * v .* r + beta*this .
232 void AddVecVec(Real alpha, const VectorBase<Real> &v,
233               const VectorBase<Real> &r, Real beta);
234
237 void AddVecDivVec(Real alpha, const VectorBase<Real> &v,
238                  const VectorBase<Real> &r, Real beta);
239
241 void Scale(Real alpha);
242
244 void MulTp(const TpMatrix<Real> &M, const MatrixTransposeType trans);
245
251 void Solve(const TpMatrix<Real> &M, const MatrixTransposeType trans);
252
254 void CopyRowsFromMat(const MatrixBase<Real> &M);
255 template<typename OtherReal>
256 void CopyRowsFromMat(const MatrixBase<OtherReal> &M);
257
259 void CopyRowsFromMat(const CuMatrixBase<Real> &M);
260
262 void CopyColsFromMat(const MatrixBase<Real> &M);
263
266 void CopyRowFromMat(const MatrixBase<Real> &M, MatrixIndexT row);
268 template<typename OtherReal>
269 void CopyRowFromMat(const MatrixBase<OtherReal> &M, MatrixIndexT row);
270
272 template<typename OtherReal>
273 void CopyRowFromSp(const SpMatrix<OtherReal> &S, MatrixIndexT row);
274
276 template<typename OtherReal>
277 void CopyColFromMat(const MatrixBase<OtherReal> &M, MatrixIndexT col);
278
280 void CopyDiagFromMat(const MatrixBase<Real> &M);
281
283 void CopyDiagFromPacked(const PackedMatrix<Real> &M);
284
285
287 inline void CopyDiagFromSp(const SpMatrix<Real> &M) { CopyDiagFromPacked(M); }
288
290 inline void CopyDiagFromTp(const TpMatrix<Real> &M) { CopyDiagFromPacked(M); }
291
293 Real Max() const;
294
297 Real Max(MatrixIndexT *index) const;
298
300 Real Min() const;
301
304 Real Min(MatrixIndexT *index) const;
305
307 Real Sum() const;
308
312 Real SumLog() const;
313
315 void AddRowSumMat(Real alpha, const MatrixBase<Real> &M, Real beta = 1.0);
316
318 void AddColSumMat(Real alpha, const MatrixBase<Real> &M, Real beta = 1.0);
319
323 void AddDiagMat2(Real alpha, const MatrixBase<Real> &M,
324                 MatrixTransposeType trans = kNoTrans, Real beta = 1.0);
325
329 void AddDiagMatMat(Real alpha, const MatrixBase<Real> &M, MatrixTransposeType
transM,
330                    const MatrixBase<Real> &N, MatrixTransposeType transN,
331                    Real beta = 1.0);

```

```

332
337 Real LogSumExp(Real prune = -1.0) const;
338
341 void Read(std::istream & in, bool binary, bool add = false);
342
344 void Write(std::ostream &Out, bool binary) const;
345
346 friend class VectorBase<double>;
347 friend class VectorBase<float>;
348 friend class CuVectorBase<Real>;
349 friend class CuVector<Real>;
350 protected:
354 ~VectorBase() {}
355
357 explicit VectorBase(): data_(NULL), dim_(0) {
358     KALDI_ASSERT_IS_FLOATING_TYPE(Real);
359 }
360
361 // Took this out since it is not currently used, and it is possible to create
362 // objects where the allocated memory is not the same size as dim_ : Arnab
363 // /// Initializer from a pointer and a size; keeps the pointer internally
364 // /// (ownership or non-ownership depends on the child class).
365 // explicit VectorBase(Real* data, MatrixIndexT dim)
366 //     : data_(data), dim_(dim) {}
367
368 // Arnab : made this protected since it is unsafe too.
370 void CopyFromPtr(const Real* Data, MatrixIndexT sz);
371
373 Real* data_;
375 MatrixIndexT dim_;
376 KALDI_DISALLOW_COPY_AND_ASSIGN(VectorBase);
377 }; // class VectorBase
378
383 template<typename Real>
384 class Vector: public VectorBase<Real> {
385 public:
387 Vector(): VectorBase<Real>() {}
388
391 explicit Vector(const MatrixIndexT s,
392                 MatrixResizeType resize_type = kSetZero)
393     : VectorBase<Real>() { Resize(s, resize_type); }
394
397 template<typename OtherReal>
398 explicit Vector(const CuVectorBase<OtherReal> &cu);
399
401 Vector(const Vector<Real> &v) : VectorBase<Real>() { // (cannot be explicit)
402     Resize(v.Dim(), kUndefined);
403     this->CopyFromVec(v);
404 }
405
407 explicit Vector(const VectorBase<Real> &v) : VectorBase<Real>() {
408     Resize(v.Dim(), kUndefined);
409     this->CopyFromVec(v);
410 }
411
413 template<typename OtherReal>
414 explicit Vector(const VectorBase<OtherReal> &v): VectorBase<Real>() {
415     Resize(v.Dim(), kUndefined);
416     this->CopyFromVec(v);
417 }
418
419 // Took this out since it is unsafe : Arnab
420 // /// Constructor from a pointer and a size; copies the data to a location
421 // /// it owns.
422 // Vector(const Real* Data, const MatrixIndexT s): VectorBase<Real>() {
423 //     Resize(s);
424 //     CopyFromPtr(Data, s);
425 // }
426
427
429 void Swap(Vector<Real> *other);
430
432 ~Vector() { Destroy(); }
433
436 void Read(std::istream & in, bool binary, bool add = false);

```

```

437
445 void Resize(MatrixIndexT length, MatrixResizeType resize_type = kSetZero);
446
448 void RemoveElement(MatrixIndexT i);
449
451 Vector<Real> &operator = (const Vector<Real> &other) {
452     Resize(other.Dim(), kUndefined);
453     this->CopyFromVec(other);
454     return *this;
455 }
456
458 Vector<Real> &operator = (const VectorBase<Real> &other) {
459     Resize(other.Dim(), kUndefined);
460     this->CopyFromVec(other);
461     return *this;
462 }
463 private:
464 void Init(const MatrixIndexT dim);
465
467 void Destroy();
468
469 };
470
472
474
475 template<typename Real>
476 class SubVector : public VectorBase<Real> {
477 public:
478     SubVector(const VectorBase<Real> &t, const MatrixIndexT origin,
479               const MatrixIndexT length) : VectorBase<Real>() {
480         // following assert equiv to origin>=0 && length>=0 &&
481         // origin+length <= rt.dim_
482         KALDI_ASSERT(static_cast<UnsignedMatrixIndexT>(origin)+
483                       static_cast<UnsignedMatrixIndexT>(length) <=
484                       static_cast<UnsignedMatrixIndexT>(t.Dim()));
485         VectorBase<Real>::data_ = const_cast<Real*>(t.Data()+origin);
486         VectorBase<Real>::dim_ = length;
487     }
488
489     SubVector(const PackedMatrix<Real> &M) {
490         VectorBase<Real>::data_ = const_cast<Real*>(M.Data());
491         VectorBase<Real>::dim_ = (M.NumRows()*(M.NumRows()+1))/2;
492     }
493
494     SubVector(const SubVector &other) : VectorBase<Real>() {
495         // this copy constructor needed for Range() to work in base class.
496         VectorBase<Real>::data_ = other.data_;
497         VectorBase<Real>::dim_ = other.dim_;
498     }
499
500     SubVector(Real *data, MatrixIndexT length) : VectorBase<Real>() {
501         VectorBase<Real>::data_ = data;
502         VectorBase<Real>::dim_ = length;
503     }
504
505     SubVector(const MatrixBase<Real> &matrix, MatrixIndexT row) {
506         VectorBase<Real>::data_ = const_cast<Real*>(matrix.RowData(row));
507         VectorBase<Real>::dim_ = matrix.NumCols();
508     }
509
510     ~SubVector() {}
511
512 private:
513     SubVector &operator = (const SubVector &other) {}
514 };
515
517 template<typename Real>
518 std::ostream & operator << (std::ostream & out, const VectorBase<Real> & v);
519
520 template<typename Real>
521 std::istream & operator >> (std::istream & in, VectorBase<Real> & v);
522
523 template<typename Real>
524 std::istream & operator >> (std::istream & in, Vector<Real> & v);
525
526

```

```

551
552
553 template<typename Real>
554 bool ApproxEqual(const VectorBase<Real> &a,
555                  const VectorBase<Real> &b, Real tol = 0.01) {
556     return a.ApproxEqual(b, tol);
557 }
558
559 template<typename Real>
560 inline void AssertEqual(VectorBase<Real> &a, VectorBase<Real> &b,
561                         float tol = 0.01) {
562     KALDI_ASSERT(a.ApproxEqual(b, tol));
563 }
564
565
566 template<typename Real>
567 Real VecVec(const VectorBase<Real> &v1, const VectorBase<Real> &v2);
568
569
570 template<typename Real, typename OtherReal>
571 Real VecVec(const VectorBase<Real> &v1, const VectorBase<OtherReal> &v2);
572
573
574 template<typename Real>
575 Real VecMatVec(const VectorBase<Real> &v1, const MatrixBase<Real> &M,
576               const VectorBase<Real> &v2);
577
578
579
580
581
582 } // namespace kald
583
584 // we need to include the implementation
585 #include "matrix/kaldi-vector-inl.h"
586
587
588
589
590 #endif // KALDI_MATRIX_KALDI_VECTOR_H_
591

```