/\* Copyright 2015 The TensorFlow Authors. All Rights Reserved.

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

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

==============================================================================\*/

// TWML modified to optimize binary features

#ifndef TENSORFLOW\_CORE\_KERNELS\_BINARY\_SPARSE\_TENSOR\_DENSE\_MATMUL\_OP\_H\_

#define TENSORFLOW\_CORE\_KERNELS\_BINARY\_SPARSE\_TENSOR\_DENSE\_MATMUL\_OP\_H\_

#include "third\_party/eigen3/unsupported/Eigen/CXX11/Tensor"

#include "tensorflow/core/framework/tensor\_types.h"

#include "tensorflow/core/framework/types.h"

#include "tensorflow/core/lib/core/errors.h"

namespace tensorflow {

namespace functor {

template <typename Device, typename T, typename Tindices, bool ADJ\_A,

bool ADJ\_B>

struct SparseTensorDenseMatMulFunctor {

static EIGEN\_ALWAYS\_INLINE Status Compute(

const Device& d, typename TTypes<T>::Matrix out,

typename TTypes<Tindices>::ConstMatrix a\_indices,

typename TTypes<T>::ConstVec a\_values, typename TTypes<T>::ConstMatrix b);

};

template <typename MATRIX, bool ADJ>

class MaybeAdjoint;

template <typename MATRIX>

class MaybeAdjoint<MATRIX, false> {

public:

EIGEN\_DEVICE\_FUNC EIGEN\_STRONG\_INLINE MaybeAdjoint(MATRIX m) : m\_(m) {}

EIGEN\_DEVICE\_FUNC EIGEN\_STRONG\_INLINE typename MATRIX::Scalar operator()(

const typename MATRIX::Index i, const typename MATRIX::Index j) const {

return m\_(i, j);

}

private:

const MATRIX m\_;

};

template <typename T>

EIGEN\_DEVICE\_FUNC EIGEN\_STRONG\_INLINE T MaybeConj(T v) {

return v;

}

template <typename MATRIX>

class MaybeAdjoint<MATRIX, true> {

public:

EIGEN\_DEVICE\_FUNC EIGEN\_STRONG\_INLINE MaybeAdjoint(MATRIX m) : m\_(m) {}

EIGEN\_DEVICE\_FUNC EIGEN\_STRONG\_INLINE typename MATRIX::Scalar operator()(

const typename MATRIX::Index i, const typename MATRIX::Index j) const {

return Eigen::numext::conj(m\_(j, i));

}

private:

const MATRIX m\_;

};

} // end namespace functor

} // end namespace tensorflow

#endif // TENSORFLOW\_CORE\_KERNELS\_BINARY\_SPARSE\_TENSOR\_DENSE\_MATMUL\_OP\_H\_