slowy07 fix: miss typo codespelling

History

7 contributors

144 lines (118 sloc) | 5.76 KB ...

```
1  /* Copyright 2017 The TensorFlow Authors. All Rights Reserved.
2
3  Licensed under the Apache License, Version 2.0 (the "License");
4  you may not use this file except in compliance with the License.
5  You may obtain a copy of the License at
6
7      http://www.apache.org/licenses/LICENSE-2.0
8
9  Unless required by applicable law or agreed to in writing, software
10 distributed under the License is distributed on an "AS IS" BASIS,
11 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 See the License for the specific language governing permissions and
13 limitations under the License.
14 ==============================================================================*/
15
16 #define EIGEN_USE_THREADS
17
18 #include "tensorflow/core/framework/op_kernel.h"
19 #include "tensorflow/core/framework/register_types.h"
20 #include "tensorflow/core/framework/tensor.h"
21 #include "tensorflow/core/framework/types.h"
22 #include "third_party/eigen3/unsupported/Eigen/CXX11/Tensor"
23
24 namespace tensorflow {
25
26 namespace {
27 template <typename T>
28 struct mod_op {
29   const T operator()(const T& a, const T& b) const { return a % b; }
```

```
};
}  // namespace

typedef Eigen::ThreadPoolDevice CPUDevice;

template <typename Tidx>
class UnravelIndexOp : public OpKernel {
 public:
  explicit UnravelIndexOp(OpKernelConstruction* ctx) : OpKernel(ctx) {}

  void Compute(OpKernelContext* ctx) override {
    const Tensor& indices_tensor = ctx->input(0);
    OP_REQUIRES(ctx,
                TensorShapeUtils::IsVector(indices_tensor.shape()) ||
                    TensorShapeUtils::IsScalar(indices_tensor.shape()),
                errors::InvalidArgument(
                    "The indices can only be scalar or vector, got \"",
                    indices_tensor.shape().DebugString(), "\""));

    const Tensor& dims_tensor = ctx->input(1);
    OP_REQUIRES(
        ctx, TensorShapeUtils::IsVector(dims_tensor.shape()),
        errors::InvalidArgument("The indices can only be 1-D, got \"",
                                dims_tensor.shape().DebugString(), "\""));

    auto dims = dims_tensor.vec<Tidx>();
    // Make sure dims does not contain a zero
    for (int i = 0; i < dims.size(); i++) {
      OP_REQUIRES(
          ctx, dims(i) != 0,
          errors::InvalidArgument("Input dims cannot contain a dim of zero, "
                                  "but dims contains zero at index ",
                                  i));
    }

    // Check to make sure indices is not out of boundary
    Eigen::Tensor<Tidx, 0, Eigen::RowMajor> dims_prod_eigen = dims.prod();
    Tidx dims_prod = dims_prod_eigen();
    const Tidx* indices = indices_tensor.flat<Tidx>().data();
    int64_t size = indices_tensor.NumElements();
    bool check = std::all_of(indices, indices + size,
                             [&](Tidx index) { return index < dims_prod; });
    OP_REQUIRES(ctx, check,
                errors::InvalidArgument("index is out of bound as with dims"));

    Eigen::array<bool, 1> reverse({true});

    Tensor strides_tensor;
    OP_REQUIRES_OK(ctx,
```

```
                  ctx->allocate_temp(DataTypeToEnum<Tidx>::value,
                                     TensorShape({dims_tensor.NumElements()}),
                                     &strides_tensor));

    auto strides = strides_tensor.vec<Tidx>();
    strides = dims.reverse(reverse)
                  .scan(0, Eigen::internal::ProdReducer<Tidx>(), false)
                  .reverse(reverse);

    Tensor strides_shifted_tensor;
    OP_REQUIRES_OK(ctx,
                   ctx->allocate_temp(DataTypeToEnum<Tidx>::value,
                                      TensorShape({dims_tensor.NumElements()}),
                                      &strides_shifted_tensor));

    auto strides_shifted = strides_shifted_tensor.vec<Tidx>();
    strides_shifted = dims.reverse(reverse)
                          .scan(0, Eigen::internal::ProdReducer<Tidx>(), true)
                          .reverse(reverse);

    Tensor* output_tensor = nullptr;
    if (TensorShapeUtils::IsScalar(indices_tensor.shape())) {
      OP_REQUIRES_OK(
          ctx, ctx->allocate_output(0, TensorShape({dims_tensor.NumElements()}),
                                    &output_tensor));

      auto output = output_tensor->vec<Tidx>();

      output = output.constant(indices_tensor.scalar<Tidx>()());
      output = output.binaryExpr(strides, mod_op<Tidx>()) / strides_shifted;
    } else {
      OP_REQUIRES_OK(
          ctx, ctx->allocate_output(0,
                                    TensorShape({dims_tensor.NumElements(),
                                                 indices_tensor.NumElements()}),
                                    &output_tensor));

      auto output = output_tensor->matrix<Tidx>();

      Eigen::array<Eigen::Index, 2> reshape{
          {static_cast<Eigen::Index>(dims_tensor.NumElements()), 1}};
      Eigen::array<Eigen::Index, 2> bcast(
          {1, static_cast<Eigen::Index>(indices_tensor.NumElements())});
      Eigen::array<Eigen::Index, 2> indices_reshape{
          {1, static_cast<Eigen::Index>(indices_tensor.NumElements())}};
      Eigen::array<Eigen::Index, 2> indices_bcast(
          {static_cast<Eigen::Index>(dims_tensor.NumElements()), 1});

      output = indices_tensor.vec<Tidx>()
```

```
                    .reshape(indices_reshape)
                    .broadcast(indices_bcast);
      output = output.binaryExpr(strides.reshape(reshape).broadcast(bcast),
                                 mod_op<Tidx>()) /
               strides_shifted.reshape(reshape).broadcast(bcast);
    }
  }
};

#define REGISTER_KERNEL(type)                                               \
  REGISTER_KERNEL_BUILDER(                                                  \
      Name("UnravelIndex").Device(DEVICE_CPU).TypeConstraint<type>("Tidx"), \
      UnravelIndexOp<type>);
TF_CALL_int32(REGISTER_KERNEL) TF_CALL_int64(REGISTER_KERNEL)
#undef REGISTER_KERNEL

}  // namespace tensorflow
```