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

#include "tensorflow/core/framework/shape\_inference.h"

#include "tensorflow/core/framework/op\_kernel.h"

#include <twml.h>

#include "tensorflow\_utils.h"

#include "resource\_utils.h"

REGISTER\_OP("DecodeAndHashBatchPredictionRequest")

.Input("input\_bytes: uint8")

.Attr("keep\_features: list(int)")

.Attr("keep\_codes: list(int)")

.Attr("decode\_mode: int = 0")

.Output("hashed\_data\_record\_handle: resource")

.SetShapeFn(shape\_inference::ScalarShape)

.Doc(R"doc(

A tensorflow OP that decodes batch prediction request and creates a handle to the batch of hashed data records.

Attr

keep\_features: a list of int ids to keep.

keep\_codes: their corresponding code.

decode\_mode: integer, indicates which decoding method to use. Let a sparse continuous

have a feature\_name and a dict of {name: value}. 0 indicates feature\_ids are computed

as hash(name). 1 indicates feature\_ids are computed as hash(feature\_name, name)

shared\_name: name used by the resource handle inside the resource manager.

container: name used by the container of the resources.

shared\_name and container are required when inheriting from ResourceOpKernel.

Input

input\_bytes: Input tensor containing the serialized batch of BatchPredictionRequest.

Outputs

hashed\_data\_record\_handle: A resource handle to the HashedDataRecordResource containing batch of HashedDataRecords.

)doc");

class DecodeAndHashBatchPredictionRequest : public OpKernel {

public:

explicit DecodeAndHashBatchPredictionRequest(OpKernelConstruction\* context)

: OpKernel(context) {

std::vector<int64> keep\_features;

std::vector<int64> keep\_codes;

OP\_REQUIRES\_OK(context, context->GetAttr("keep\_features", &keep\_features));

OP\_REQUIRES\_OK(context, context->GetAttr("keep\_codes", &keep\_codes));

OP\_REQUIRES\_OK(context, context->GetAttr("decode\_mode", &m\_decode\_mode));

OP\_REQUIRES(context, keep\_features.size() == keep\_codes.size(),

errors::InvalidArgument("keep keys and values must have same size."));

#ifdef USE\_DENSE\_HASH

m\_keep\_map.set\_empty\_key(0);

#endif // USE\_DENSE\_HASH

for (uint64\_t i = 0; i < keep\_features.size(); i++) {

m\_keep\_map[keep\_features[i]] = keep\_codes[i];

}

}

private:

twml::Map<int64\_t, int64\_t> m\_keep\_map;

int64 m\_decode\_mode;

void Compute(OpKernelContext\* context) override {

try {

HashedDataRecordResource \*resource = nullptr;

OP\_REQUIRES\_OK(context, makeResourceHandle<HashedDataRecordResource>(context, 0, &resource));

// Store the input bytes in the resource so it isnt freed before the resource.

// This is necessary because we are not copying the contents for tensors.

resource->input = context->input(0);

const uint8\_t \*input\_bytes = resource->input.flat<uint8>().data();

twml::HashedDataRecordReader reader;

twml::HashedBatchPredictionRequest bpr;

reader.setKeepMap(&m\_keep\_map);

reader.setBuffer(input\_bytes);

reader.setDecodeMode(m\_decode\_mode);

bpr.decode(reader);

resource->common = std::move(bpr.common());

resource->records = std::move(bpr.requests());

// Each datarecord has a copy of common features.

// Initialize total\_size by common\_size \* num\_records

int64 common\_size = static\_cast<int64>(resource->common.totalSize());

int64 num\_records = static\_cast<int64>(resource->records.size());

int64 total\_size = common\_size \* num\_records;

for (const auto &record : resource->records) {

total\_size += static\_cast<int64>(record.totalSize());

}

resource->total\_size = total\_size;

resource->num\_labels = 0;

resource->num\_weights = 0;

} catch (const std::exception &e) {

context->CtxFailureWithWarning(errors::InvalidArgument(e.what()));

}

}

};

REGISTER\_KERNEL\_BUILDER(

Name("DecodeAndHashBatchPredictionRequest").Device(DEVICE\_CPU),

DecodeAndHashBatchPredictionRequest);

REGISTER\_OP("DecodeBatchPredictionRequest")

.Input("input\_bytes: uint8")

.Attr("keep\_features: list(int)")

.Attr("keep\_codes: list(int)")

.Output("data\_record\_handle: resource")

.SetShapeFn(shape\_inference::ScalarShape)

.Doc(R"doc(

A tensorflow OP that decodes batch prediction request and creates a handle to the batch of data records.

Attr

keep\_features: a list of int ids to keep.

keep\_codes: their corresponding code.

shared\_name: name used by the resource handle inside the resource manager.

container: name used by the container of the resources.

shared\_name and container are required when inheriting from ResourceOpKernel.

Input

input\_bytes: Input tensor containing the serialized batch of BatchPredictionRequest.

Outputs

data\_record\_handle: A resource handle to the DataRecordResource containing batch of DataRecords.

)doc");

class DecodeBatchPredictionRequest : public OpKernel {

public:

explicit DecodeBatchPredictionRequest(OpKernelConstruction\* context)

: OpKernel(context) {

std::vector<int64> keep\_features;

std::vector<int64> keep\_codes;

OP\_REQUIRES\_OK(context, context->GetAttr("keep\_features", &keep\_features));

OP\_REQUIRES\_OK(context, context->GetAttr("keep\_codes", &keep\_codes));

OP\_REQUIRES(context, keep\_features.size() == keep\_codes.size(),

errors::InvalidArgument("keep keys and values must have same size."));

#ifdef USE\_DENSE\_HASH

m\_keep\_map.set\_empty\_key(0);

#endif // USE\_DENSE\_HASH

for (uint64\_t i = 0; i < keep\_features.size(); i++) {

m\_keep\_map[keep\_features[i]] = keep\_codes[i];

}

}

private:

twml::Map<int64\_t, int64\_t> m\_keep\_map;

void Compute(OpKernelContext\* context) override {

try {

DataRecordResource \*resource = nullptr;

OP\_REQUIRES\_OK(context, makeResourceHandle<DataRecordResource>(context, 0, &resource));

// Store the input bytes in the resource so it isnt freed before the resource.

// This is necessary because we are not copying the contents for tensors.

resource->input = context->input(0);

const uint8\_t \*input\_bytes = resource->input.flat<uint8>().data();

twml::DataRecordReader reader;

twml::BatchPredictionRequest bpr;

reader.setKeepMap(&m\_keep\_map);

reader.setBuffer(input\_bytes);

bpr.decode(reader);

resource->common = std::move(bpr.common());

resource->records = std::move(bpr.requests());

resource->num\_weights = 0;

resource->num\_labels = 0;

resource->keep\_map = &m\_keep\_map;

} catch (const std::exception &e) {

context->CtxFailureWithWarning(errors::InvalidArgument(e.what()));

}

}

};

REGISTER\_KERNEL\_BUILDER(

Name("DecodeBatchPredictionRequest").Device(DEVICE\_CPU),

DecodeBatchPredictionRequest);