#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"

#include <functional>

REGISTER\_OP("DecodeAndHashDataRecord")

.Attr("InputType: {uint8, string}")

.Input("input\_bytes: InputType")

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

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

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

.Attr("weight\_features: list(int) = []")

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

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

.SetShapeFn(shape\_inference::ScalarShape)

.Doc(R"doc(

A tensorflow OP that creates a handle for the hashed data record.

Attr

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

keep\_codes: their corresponding code.

label\_features: list of feature ids representing the labels.

weight\_features: list of feature ids representing the weights. Defaults to empty list.

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.

Input

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

Outputs

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

)doc");

template<typename InputType>

class DecodeAndHashDataRecord : public OpKernel {

public:

explicit DecodeAndHashDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {

std::vector<int64> keep\_features;

std::vector<int64> keep\_codes;

std::vector<int64> label\_features;

std::vector<int64> weight\_features;

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("label\_features", &label\_features));

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

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);

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

m\_weights\_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];

}

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

m\_labels\_map[label\_features[i]] = i;

}

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

m\_weights\_map[weight\_features[i]] = i;

}

}

private:

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

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

twml::Map<int64\_t, int64\_t> m\_weights\_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);

int batch\_size = getBatchSize<InputType>(resource->input);

int num\_labels = static\_cast<int>(m\_labels\_map.size());

int num\_weights = static\_cast<int>(m\_weights\_map.size());

twml::HashedDataRecordReader reader;

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

reader.setLabelsMap(&m\_labels\_map);

reader.setDecodeMode(m\_decode\_mode);

// Do not set weight map if it is empty. This will take a faster path.

if (num\_weights != 0) {

reader.setWeightsMap(&m\_weights\_map);

}

resource->records.clear();

resource->records.reserve(batch\_size);

int64 total\_size = 0;

for (int id = 0; id < batch\_size; id++) {

const uint8\_t \*input\_bytes = getInputBytes<InputType>(resource->input, id);

reader.setBuffer(input\_bytes);

resource->records.emplace\_back(num\_labels, num\_weights);

resource->records[id].decode(reader);

total\_size += static\_cast<int64>(resource->records[id].totalSize());

}

resource->total\_size = total\_size;

resource->num\_labels = num\_labels;

resource->num\_weights = num\_weights;

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

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

}

}

};

REGISTER\_OP("GetIdsFromHashedDataRecord")

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

.Output("ids: int64")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns unhashed ids from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

ids: ids specifies the index of the records[id] in the batch (int64)

)doc");

// This Kernel is used for both training and serving once the resource is created.

class GetIdsFromHashedDataRecord : public OpKernel {

public:

explicit GetIdsFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

const auto &records = handle->records;

const auto &common = handle->common;

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

const int64 total\_size = handle->total\_size;

TensorShape shape = {total\_size};

Tensor \*ids;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, shape, &ids));

int id = 0;

int64 offset = 0;

auto ids\_flat = ids->flat<int64>();

for (const auto &record : records) {

// Since common features are added to each input, add the common\_size to the current size.

// For training common\_size == 0, for serving it can be a non-zero value.

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

std::fill(ids\_flat.data() + offset, ids\_flat.data() + offset + curr\_size, id);

offset += curr\_size;

id++;

}

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

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

}

}

};

// OutType: Output Tensor Type. FieldType: The storage type used inside HashedDatarecord.

template<typename OutType, typename FieldType>

class GetOutputFromHashedDataRecord : public OpKernel {

protected:

using Getter = std::function<const std::vector<FieldType>&(const twml::HashedDataRecord &)>;

Getter getter;

public:

explicit GetOutputFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

const auto &records = handle->records;

const auto &common = handle->common;

const int64 total\_size = handle->total\_size;

TensorShape shape = {total\_size};

Tensor \*output;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, shape, &output));

const auto &common\_output = getter(common);

auto output\_data = output->flat<OutType>().data();

for (const auto &record : records) {

// This is does not copy anything during training as common\_size == 0

// It will copy the relevant common features coming from a batch prediction request.

output\_data = std::copy(common\_output.begin(), common\_output.end(), output\_data);

// Copy the current record to output.

const auto& rec\_output = getter(record);

output\_data = std::copy(rec\_output.begin(), rec\_output.end(), output\_data);

}

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

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

}

}

};

REGISTER\_OP("GetUKeysFromHashedDataRecord")

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

.Output("ukeys: int64")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns unhashed keys from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

ukeys: unhased keys / raw feature ids from the original request.

)doc");

class GetUKeysFromHashedDataRecord : public GetOutputFromHashedDataRecord<int64, int64\_t> {

public:

explicit GetUKeysFromHashedDataRecord(OpKernelConstruction\* context)

: GetOutputFromHashedDataRecord<int64, int64\_t>(context){

getter = [](const twml::HashedDataRecord &record) -> const std::vector<int64\_t> & {

return record.keys();

};

}

};

REGISTER\_OP("GetKeysFromHashedDataRecord")

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

.Output("keys: int64")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns keys from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

keys: keys after raw feature ids are hashed with values (int64)

)doc");

class GetKeysFromHashedDataRecord : public GetOutputFromHashedDataRecord<int64, int64\_t> {

public:

explicit GetKeysFromHashedDataRecord(OpKernelConstruction\* context)

: GetOutputFromHashedDataRecord<int64, int64\_t>(context){

getter = [](const twml::HashedDataRecord &record) -> const std::vector<int64\_t> & {

return record.transformed\_keys();

};

}

};

REGISTER\_OP("GetValuesFromHashedDataRecord")

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

.Output("values: float")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns values from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

values: feature values.

)doc");

class GetValuesFromHashedDataRecord : public GetOutputFromHashedDataRecord<float, double> {

public:

explicit GetValuesFromHashedDataRecord(OpKernelConstruction\* context)

: GetOutputFromHashedDataRecord<float, double>(context){

getter = [](const twml::HashedDataRecord &record) -> const std::vector<double> & {

return record.values();

};

}

};

REGISTER\_OP("GetCodesFromHashedDataRecord")

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

.Output("codes: int64")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns codes from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

codes: deepbird feature code, usually from A,B,C,D ... in the config.

)doc");

class GetCodesFromHashedDataRecord : public GetOutputFromHashedDataRecord<int64, int64\_t> {

public:

explicit GetCodesFromHashedDataRecord(OpKernelConstruction\* context)

: GetOutputFromHashedDataRecord<int64, int64\_t>(context){

getter = [](const twml::HashedDataRecord &record) -> const std::vector<int64\_t> & {

return record.codes();

};

}

};

REGISTER\_OP("GetTypesFromHashedDataRecord")

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

.Output("types: int8")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns types from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

types: feature types corresponding to BINARY, DISCRETE, etc.

)doc");

class GetTypesFromHashedDataRecord : public GetOutputFromHashedDataRecord<int8, uint8\_t> {

public:

explicit GetTypesFromHashedDataRecord(OpKernelConstruction\* context)

: GetOutputFromHashedDataRecord<int8, uint8\_t>(context){

getter = [](const twml::HashedDataRecord &record) -> const std::vector<uint8\_t> & {

return record.types();

};

}

};

REGISTER\_OP("GetBatchSizeFromHashedDataRecord")

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

.Output("batch\_size: int64")

.SetShapeFn(shape\_inference::ScalarShape)

.Doc(R"doc(

A tensorflow OP that returns batch size from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

batch\_size: Number of records held in the handle.

)doc");

class GetBatchSizeFromHashedDataRecord : public OpKernel {

public:

explicit GetBatchSizeFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

Tensor \*output;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, TensorShape({}), &output));

output->scalar<int64>()() = handle->records.size();

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

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

}

}

};

REGISTER\_OP("GetTotalSizeFromHashedDataRecord")

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

.Output("total\_size: int64")

.SetShapeFn(shape\_inference::ScalarShape)

.Doc(R"doc(

A tensorflow OP that returns total size from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

total\_size: Total number of keys / values in the batch.

)doc");

class GetTotalSizeFromHashedDataRecord : public OpKernel {

public:

explicit GetTotalSizeFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

Tensor \*output;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, TensorShape({}), &output));

output->scalar<int64>()() = handle->total\_size;

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

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

}

}

};

REGISTER\_OP("GetLabelsFromHashedDataRecord")

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

.Output("labels: float")

.Attr("default\_label: float")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns labels from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

labels: A 2D tensor of size [batch\_size, num\_labels] containing the label values.

)doc");

class GetLabelsFromHashedDataRecord : public OpKernel {

private:

float default\_label;

public:

explicit GetLabelsFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {

OP\_REQUIRES\_OK(context, context->GetAttr("default\_label", &default\_label));

}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

const auto &records = handle->records;

const int num\_labels = static\_cast<int>(handle->num\_labels);

TensorShape shape = {static\_cast<int64>(handle->records.size()), num\_labels};

Tensor \*labels;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, shape, &labels));

// The default value of label is not present in data record is std::nanf

// For continuous labels, change that to a default\_label or label.

auto func = [this](float label) -> float {

return std::isnan(label) ? default\_label : label;

};

auto labels\_data = labels->flat<float>().data();

for (const auto &record : records) {

const auto& rec\_labels = record.labels();

labels\_data = std::transform(rec\_labels.begin(), rec\_labels.end(), labels\_data, func);

}

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

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

}

}

};

REGISTER\_OP("GetWeightsFromHashedDataRecord")

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

.Output("weights: float")

.SetShapeFn([](::tensorflow::shape\_inference::InferenceContext\* c) {

return Status::OK();

}).Doc(R"doc(

A tensorflow OP that returns weights from the hashed data record.

Input

hashed\_data\_record\_handle: Resource handle to DataRecord

Outputs

weights: A 2D tensor of size [batch\_size, num\_weights] containing the weight values.

)doc");

class GetWeightsFromHashedDataRecord : public OpKernel {

public:

explicit GetWeightsFromHashedDataRecord(OpKernelConstruction\* context)

: OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

auto handle = getHandle<HashedDataRecordResource>(context, 0);

const auto &records = handle->records;

const int num\_weights = static\_cast<int>(handle->num\_weights);

TensorShape shape = {static\_cast<int64>(handle->records.size()), num\_weights};

Tensor \*weights;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, shape, &weights));

auto weights\_data = weights->flat<float>().data();

for (const auto &record : records) {

const auto& rec\_weights = record.weights();

weights\_data = std::copy(rec\_weights.begin(), rec\_weights.end(), weights\_data);

}

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

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

}

}

};

#define REGISTER\_DECODE\_AND\_HASH(InputType) \

REGISTER\_KERNEL\_BUILDER( \

Name("DecodeAndHashDataRecord") \

.Device(DEVICE\_CPU) \

.TypeConstraint<InputType>("InputType"), \

DecodeAndHashDataRecord<InputType>); \

REGISTER\_DECODE\_AND\_HASH(uint8)

REGISTER\_DECODE\_AND\_HASH(string)

#define REGISTER\_GETTER(FIELD) \

REGISTER\_KERNEL\_BUILDER( \

Name("Get" #FIELD "FromHashedDataRecord") \

.Device(DEVICE\_CPU), \

Get##FIELD##FromHashedDataRecord); \

REGISTER\_GETTER(Ids)

REGISTER\_GETTER(UKeys)

REGISTER\_GETTER(Keys)

REGISTER\_GETTER(Values)

REGISTER\_GETTER(Codes)

REGISTER\_GETTER(Types)

REGISTER\_GETTER(BatchSize)

REGISTER\_GETTER(TotalSize)

REGISTER\_GETTER(Labels)

REGISTER\_GETTER(Weights)