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

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

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

#include <twml.h>

#include <mutex>

using namespace tensorflow;

REGISTER\_OP("Hashmap")

.Input("keys: int64")

.Input("hash\_keys: int64")

.Input("hash\_values: int64")

.Output("values: int64")

.Output("mask: int8")

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

// TODO: check if the sizes are different in the input

c->set\_output(0, c->input(0));

c->set\_output(1, c->input(0));

return Status::OK();

});

class Hashmap : public OpKernel {

private:

twml::HashMap hmap;

std::once\_flag flag;

public:

explicit Hashmap(OpKernelConstruction\* context) : OpKernel(context) {}

void Compute(OpKernelContext\* context) override {

try {

// Quick hack

const Tensor& keys = context->input(0);

std::call\_once(this->flag, [this, context](){

const Tensor& hash\_keys = context->input(1);

const Tensor& hash\_values = context->input(2);

const auto hash\_keys\_flat = hash\_keys.flat<int64>();

const auto hash\_values\_flat = hash\_values.flat<int64>();

const int64 N = hash\_keys\_flat.size();

for (int64 i = 0; i < N; i++) {

hmap.insert(hash\_keys\_flat(i), hash\_values\_flat(i));

}

});

Tensor\* values = nullptr;

OP\_REQUIRES\_OK(context, context->allocate\_output(0, keys.shape(),

&values));

Tensor\* mask = nullptr;

OP\_REQUIRES\_OK(context, context->allocate\_output(1, keys.shape(),

&mask));

// copy the values without sharing a storage

values->flat<int64>() = keys.flat<int64>();

auto keys\_flat = keys.flat<int64>();

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

auto mask\_flat = mask->flat<int8>();

// TODO: use twml tensor

const int64 N = keys\_flat.size();

for (int64 i = 0; i < N; i++) {

// values\_flat(i), keys\_flat(i) return references to tensorflow::int64.

// Using them in hmap.get() was causing issues because of automatic casting.

int64\_t val = values\_flat(i);

int64\_t key = keys\_flat(i);

mask\_flat(i) = hmap.get(val, key);

values\_flat(i) = val;

}

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

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

}

}

};

REGISTER\_KERNEL\_BUILDER(

Name("Hashmap")

.Device(DEVICE\_CPU),

Hashmap);