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

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

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

#include <algorithm> // std::fill\_n

using namespace tensorflow;

REGISTER\_OP("CompressSampleIds")

.Attr("T: {int32}")

.Input("input: T")

.Output("output: T")

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

c->set\_output(0, c->Vector(c->kUnknownDim));

return Status::OK();

});

template<typename T>

class CompressSampleIds : public OpKernel {

public:

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

void Compute(OpKernelContext\* context) override {

// Grab the input tensor

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

auto input = input\_tensor.flat<T>();

const int N = input.size();

// Check for improper input

bool error = (N > 0 && input(0) < 0);

for (int i = 1; !error && i < N; i++) {

error = input(i - 1) > input(i);

}

OP\_REQUIRES(

context, !error,

errors::InvalidArgument(

"Error in CompressSampleIds. SampleIds must be non-negative and non-decreasing"

)

);

// choose output size, either last input element + 1, or 0

int output\_size = 0;

if (N > 0) {

output\_size = input(N - 1) + 1;

}

// Create an output tensor

Tensor\* output\_tensor = nullptr;

OP\_REQUIRES\_OK(

context,

context->allocate\_output(0, TensorShape({output\_size}), &output\_tensor)

);

auto output\_flat = output\_tensor->flat<T>();

// Zero-initialize output

for (int i = 0; i < output\_size; i++) {

output\_flat(i) = 0;

}

// count how many of each input element

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

output\_flat(input(i)) ++;

}

}

};

REGISTER\_OP("DecompressSampleIds")

.Attr("T: {int32}")

.Input("input: T")

.Output("output: T")

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

c->set\_output(0, c->Vector(c->kUnknownDim));

return Status::OK();

});

template<typename T>

class DecompressSampleIds : public OpKernel {

public:

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

void Compute(OpKernelContext\* context) override {

// Grab the input tensor

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

auto input = input\_tensor.flat<T>();

const int N = input.size();

// Check for improper input

bool error = false;

int output\_size = 0;

for (int i = 0; !error && i < N; i++) {

error = input(i) < 0;

output\_size += input(i);

}

OP\_REQUIRES(

context, !error,

errors::InvalidArgument(

"Error in DecompressSampleIds. Inputs must be non-negative."

)

);

// Create an output tensor

Tensor\* output\_tensor = nullptr;

OP\_REQUIRES\_OK(

context,

context->allocate\_output(0, TensorShape({output\_size}),&output\_tensor)

);

auto output\_flat = output\_tensor->flat<T>();

T \*output\_data = output\_flat.data();

for (int current\_sample = 0; current\_sample < N; current\_sample++) {

std::fill\_n(output\_data, input(current\_sample), current\_sample);

output\_data += input(current\_sample);

}

}

};

#define REGISTER(Type) \

\

REGISTER\_KERNEL\_BUILDER( \

Name("CompressSampleIds") \

.Device(DEVICE\_CPU) \

.TypeConstraint<Type>("T"), \

CompressSampleIds<Type>); \

\

REGISTER\_KERNEL\_BUILDER( \

Name("DecompressSampleIds") \

.Device(DEVICE\_CPU) \

.TypeConstraint<Type>("T"), \

DecompressSampleIds<Type>); \

\

REGISTER(int32);