#include "internal/interpolate.h"

#include "internal/error.h"

#include <twml/discretizer\_impl.h>

#include <twml/optim.h>

namespace twml {

// it is assumed that start\_compute and end\_compute are valid

template<typename T>

void discretizerInfer(Tensor &output\_keys,

Tensor &output\_vals,

const Tensor &input\_ids,

const Tensor &input\_vals,

const Tensor &bin\_ids,

const Tensor &bin\_vals,

const Tensor &feature\_offsets,

int output\_bits,

const Map<int64\_t, int64\_t> &ID\_to\_index,

int64\_t start\_compute,

int64\_t end\_compute,

int64\_t output\_start) {

auto out\_keysData = output\_keys.getData<int64\_t>();

auto out\_valsData = output\_vals.getData<T>();

uint64\_t out\_keysStride = output\_keys.getStride(0);

uint64\_t out\_valsStride = output\_vals.getStride(0);

auto in\_idsData = input\_ids.getData<int64\_t>();

auto in\_valsData = input\_vals.getData<T>();

uint64\_t in\_idsStride = input\_ids.getStride(0);

uint64\_t in\_valsStride = input\_vals.getStride(0);

auto xsData = bin\_vals.getData<T>();

auto ysData = bin\_ids.getData<int64\_t>();

uint64\_t xsStride = bin\_vals.getStride(0);

uint64\_t ysStride = bin\_ids.getStride(0);

auto offsetData = feature\_offsets.getData<int64\_t>();

uint64\_t total\_bins = bin\_ids.getNumElements();

uint64\_t fsize = feature\_offsets.getNumElements();

uint64\_t output\_size = (1 << output\_bits);

for (uint64\_t i = start\_compute; i < end\_compute; i++) {

int64\_t feature\_ID = in\_idsData[i \* in\_idsStride];

T val = in\_valsData[i \* in\_valsStride];

auto iter = ID\_to\_index.find(feature\_ID);

if (iter == ID\_to\_index.end()) {

// feature not calibrated

// modulo add operation for new key from feature ID

int64\_t ikey = feature\_ID % (output\_size - total\_bins) + total\_bins;

out\_keysData[(i + output\_start - start\_compute) \* out\_keysStride] = ikey;

out\_valsData[(i + output\_start - start\_compute) \* out\_valsStride] = val;

continue;

}

int64\_t ikey = iter->second;

// Perform interpolation

uint64\_t offset = offsetData[ikey];

uint64\_t next\_offset = (ikey == (int64\_t)(fsize - 1)) ? total\_bins : offsetData[ikey + 1];

uint64\_t mainSize = next\_offset - offset;

const T \*lxsData = xsData + offset;

const int64\_t \*lysData = ysData + offset;

int64\_t okey;

okey = interpolation<T, int64\_t>(lxsData, xsStride,

lysData, ysStride,

val, mainSize,

NEAREST, 0);

out\_keysData[(i + output\_start - start\_compute) \* out\_keysStride] = okey;

out\_valsData[(i + output\_start - start\_compute) \* out\_valsStride] = 1;

}

}

void discretizerInfer(Tensor &output\_keys,

Tensor &output\_vals,

const Tensor &input\_ids,

const Tensor &input\_vals,

const Tensor &bin\_ids,

const Tensor &bin\_vals,

const Tensor &feature\_offsets,

int output\_bits,

const Map<int64\_t, int64\_t> &ID\_to\_index,

int start\_compute,

int end\_compute,

int output\_start) {

if (input\_ids.getType() != TWML\_TYPE\_INT64) {

throw twml::Error(TWML\_ERR\_TYPE, "input\_ids must be a Long Tensor");

}

if (output\_keys.getType() != TWML\_TYPE\_INT64) {

throw twml::Error(TWML\_ERR\_TYPE, "output\_keys must be a Long Tensor");

}

if (bin\_ids.getType() != TWML\_TYPE\_INT64) {

throw twml::Error(TWML\_ERR\_TYPE, "bin\_ids must be a Long Tensor");

}

if (feature\_offsets.getType() != TWML\_TYPE\_INT64) {

throw twml::Error(TWML\_ERR\_TYPE, "bin\_ids must be a Long Tensor");

}

if (input\_vals.getType() != bin\_vals.getType()) {

throw twml::Error(TWML\_ERR\_TYPE,

"Data type of input\_vals does not match type of bin\_vals");

}

if (bin\_vals.getNumDims() != 1) {

throw twml::Error(TWML\_ERR\_SIZE,

"bin\_vals must be 1 Dimensional");

}

if (bin\_ids.getNumDims() != 1) {

throw twml::Error(TWML\_ERR\_SIZE,

"bin\_ids must be 1 Dimensional");

}

if (bin\_vals.getNumElements() != bin\_ids.getNumElements()) {

throw twml::Error(TWML\_ERR\_SIZE,

"Dimensions of bin\_vals and bin\_ids do not match");

}

if (feature\_offsets.getStride(0) != 1) {

throw twml::Error(TWML\_ERR\_SIZE,

"feature\_offsets must be contiguous");

}

uint64\_t size = input\_ids.getDim(0);

if (end\_compute == -1) {

end\_compute = size;

}

if (start\_compute < 0 || start\_compute >= size) {

throw twml::Error(TWML\_ERR\_SIZE,

"start\_compute out of range");

}

if (end\_compute < -1 || end\_compute > size) {

throw twml::Error(TWML\_ERR\_SIZE,

"end\_compute out of range");

}

if (start\_compute > end\_compute && end\_compute != -1) {

throw twml::Error(TWML\_ERR\_SIZE,

"must have start\_compute <= end\_compute, or end\_compute==-1");

}

switch (input\_vals.getType()) {

case TWML\_TYPE\_FLOAT:

twml::discretizerInfer<float>(output\_keys, output\_vals,

input\_ids, input\_vals,

bin\_ids, bin\_vals, feature\_offsets, output\_bits, ID\_to\_index,

start\_compute, end\_compute, output\_start);

break;

case TWML\_TYPE\_DOUBLE:

twml::discretizerInfer<double>(output\_keys, output\_vals,

input\_ids, input\_vals,

bin\_ids, bin\_vals, feature\_offsets, output\_bits, ID\_to\_index,

start\_compute, end\_compute, output\_start);

break;

default:

throw twml::Error(TWML\_ERR\_TYPE,

"Unsupported datatype for discretizerInfer");

}

}

} // namespace twml