#include "internal/interpolate.h"

#include "internal/error.h"

#include <twml/optim.h>

namespace twml {

template<typename T>

void mdlInfer(Tensor &output\_keys, Tensor &output\_vals,

const Tensor &input\_keys, const Tensor &input\_vals,

const Tensor &bin\_ids,

const Tensor &bin\_vals,

const Tensor &feature\_offsets,

bool return\_bin\_indices) {

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

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

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

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

auto ikeysData = input\_keys.getData<int64\_t>();

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

uint64\_t ikeysStride = input\_keys.getStride(0);

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

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

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

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

int64\_t ikey = ikeysData[i \* ikeysStride] - TWML\_INDEX\_BASE;

T val = ivalsData[i \* ivaluesStride];

if (ikey == -1) {

ovalsData[i \* ovaluesStride] = val;

continue;

}

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

lysData, ysStride,

val, mainSize, NEAREST, 0,

return\_bin\_indices);

okeysData[i \* okeysStride] = okey + TWML\_INDEX\_BASE;

ovalsData[i \* ovaluesStride] = 1;

}

}

void mdlInfer(Tensor &output\_keys, Tensor &output\_vals,

const Tensor &input\_keys, const Tensor &input\_vals,

const Tensor &bin\_ids,

const Tensor &bin\_vals,

const Tensor &feature\_offsets,

bool return\_bin\_indices) {

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

throw twml::Error(TWML\_ERR\_TYPE, "input\_keys 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");

}

switch (input\_vals.getType()) {

case TWML\_TYPE\_FLOAT:

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

input\_keys, input\_vals,

bin\_ids, bin\_vals, feature\_offsets,

return\_bin\_indices);

break;

case TWML\_TYPE\_DOUBLE:

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

input\_keys, input\_vals,

bin\_ids, bin\_vals, feature\_offsets,

return\_bin\_indices);

break;

default:

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

"Unsupported datatype for mdlInfer");

}

}

const int DEFAULT\_INTERPOLATION\_LOWEST = 0;

/\*\*

\* @param output tensor to hold linear or nearest interpolation output.

\* This function does not allocate space.

\* The output tensor must have space allcoated.

\* @param input input tensor; size must match output.

\* input is assumed to have size [batch\_size, number\_of\_labels].

\* @param xs the bins.

\* @param ys the values for the bins.

\* @param mode: linear or nearest InterpolationMode.

\* linear is used for isotonic calibration.

\* nearest is used for MDL calibration and MDL inference.

\*

\* @return Returns nothing. Output is stored into the output tensor.

\*

\* This is used by IsotonicCalibration inference.

\*/

template <typename T>

void interpolation(

Tensor output,

const Tensor input,

const Tensor xs,

const Tensor ys,

const InterpolationMode mode) {

// Sanity check: input and output should have two dims.

if (input.getNumDims() != 2 || output.getNumDims() != 2) {

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

"input and output should have 2 dimensions.");

}

// Sanity check: input and output size should match.

for (int i = 0; i < input.getNumDims(); i++) {

if (input.getDim(i) != output.getDim(i)) {

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

"input and output mismatch in size.");

}

}

// Sanity check: number of labels in input should match

// number of labels in xs / ys.

if (input.getDim(1) != xs.getDim(0)

|| input.getDim(1) != ys.getDim(0)) {

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

"input, xs, ys should have the same number of labels.");

}

const uint64\_t inputStride0 = input.getStride(0);

const uint64\_t inputStride1 = input.getStride(1);

const uint64\_t outputStride0 = output.getStride(0);

const uint64\_t outputStride1 = output.getStride(1);

const uint64\_t xsStride0 = xs.getStride(0);

const uint64\_t xsStride1 = xs.getStride(1);

const uint64\_t ysStride0 = ys.getStride(0);

const uint64\_t ysStride1 = ys.getStride(1);

const uint64\_t mainSize = xs.getDim(1);

// for each value in the input matrix, compute output value by

// calling interpolation.

auto inputData = input.getData<T>();

auto outputData = output.getData<T>();

auto xsData = xs.getData<T>();

auto ysData = ys.getData<T>();

for (uint64\_t i = 0; i < input.getDim(0); i++) {

for (uint64\_t j = 0; j < input.getDim(1); j++) {

const T val = inputData[i \* inputStride0 + j \* inputStride1];

const T \*lxsData = xsData + j \* xsStride0;

const T \*lysData = ysData + j \* ysStride0;

const T res = interpolation(

lxsData, xsStride1,

lysData, ysStride1,

val,

mainSize,

mode,

DEFAULT\_INTERPOLATION\_LOWEST);

outputData[i \* outputStride0 + j \* outputStride1] = res;

}

}

}

void linearInterpolation(

Tensor output,

const Tensor input,

const Tensor xs,

const Tensor ys) {

switch (input.getType()) {

case TWML\_TYPE\_FLOAT:

twml::interpolation<float>(output, input, xs, ys, LINEAR);

break;

case TWML\_TYPE\_DOUBLE:

twml::interpolation<double>(output, input, xs, ys, LINEAR);

break;

default:

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

"Unsupported datatype for linearInterpolation.");

}

}

void nearestInterpolation(

Tensor output,

const Tensor input,

const Tensor xs,

const Tensor ys) {

switch (input.getType()) {

case TWML\_TYPE\_FLOAT:

twml::interpolation<float>(output, input, xs, ys, NEAREST);

break;

case TWML\_TYPE\_DOUBLE:

twml::interpolation<double>(output, input, xs, ys, NEAREST);

break;

default:

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

"Unsupported datatype for nearestInterpolation.");

}

}

} // namespace twml

twml\_err twml\_optim\_mdl\_infer(twml\_tensor output\_keys,

twml\_tensor output\_vals,

const twml\_tensor input\_keys,

const twml\_tensor input\_vals,

const twml\_tensor bin\_ids,

const twml\_tensor bin\_vals,

const twml\_tensor feature\_offsets,

bool return\_bin\_indices) {

HANDLE\_EXCEPTIONS(

using namespace twml;

mdlInfer(\*getTensor(output\_keys),

\*getTensor(output\_vals),

\*getConstTensor(input\_keys),

\*getConstTensor(input\_vals),

\*getConstTensor(bin\_ids),

\*getConstTensor(bin\_vals),

\*getConstTensor(feature\_offsets),

return\_bin\_indices););

return TWML\_ERR\_NONE;

}

twml\_err twml\_optim\_nearest\_interpolation(

twml\_tensor output,

const twml\_tensor input,

const twml\_tensor xs,

const twml\_tensor ys) {

HANDLE\_EXCEPTIONS(

using namespace twml;

nearestInterpolation(\*getTensor(output),

\*getConstTensor(input),

\*getConstTensor(xs),

\*getConstTensor(ys)););

return TWML\_ERR\_NONE;

}