#include "internal/error.h"

#include "internal/murmur\_hash3.h"

#include "internal/utf\_converter.h"

#include <twml/functions.h>

#include <cstring>

#include <algorithm>

namespace twml {

template<typename T>

void add1(Tensor &output, const Tensor input) {

T \*odata = output.getData<T>();

const T \*idata = input.getData<T>();

const uint64\_t num\_elements = input.getNumElements();

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

odata[i] = idata[i] + 1;

}

}

template<typename T>

void copy(Tensor &output, const Tensor input) {

T \*odata = output.getData<T>();

const T \*idata = input.getData<T>();

const uint64\_t num\_elements = input.getNumElements();

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

odata[i] = idata[i];

}

}

void add1(Tensor &output, const Tensor input) {

auto type = input.getType();

if (output.getType() != type) {

throw twml::Error(TWML\_ERR\_TYPE, "Output type does not match input type");

}

if (output.getNumElements() != input.getNumElements()) {

throw twml::Error(TWML\_ERR\_SIZE, "Output size does not match input size");

}

// TODO: Implement an easier dispatch function

switch (type) {

case TWML\_TYPE\_FLOAT:

twml::add1<float>(output, input);

break;

case TWML\_TYPE\_DOUBLE:

twml::add1<double>(output, input);

break;

default:

throw twml::Error(TWML\_ERR\_TYPE, "add1 only supports float and double tensors");

}

}

void copy(Tensor &output, const Tensor input) {

auto type = input.getType();

if (output.getType() != type) {

throw twml::Error(TWML\_ERR\_TYPE, "Output type does not match input type");

}

if (output.getNumElements() != input.getNumElements()) {

throw twml::Error(TWML\_ERR\_SIZE, "Output size does not match input size");

}

// TODO: Implement an easier dispatch function

switch (type) {

case TWML\_TYPE\_FLOAT:

twml::copy<float>(output, input);

break;

case TWML\_TYPE\_DOUBLE:

twml::copy<double>(output, input);

break;

default:

throw twml::Error(TWML\_ERR\_TYPE, "copy only supports float and double tensors");

}

}

int64\_t featureId(const std::string &feature) {

const char \*str = feature.c\_str();

uint64\_t len = feature.size();

int64\_t id = 0;

TWML\_CHECK(twml\_get\_feature\_id(&id, len, str), "Error getting featureId");

return id;

}

} // namespace twml

twml\_err twml\_add1(twml\_tensor output, const twml\_tensor input) {

HANDLE\_EXCEPTIONS(

auto out = twml::getTensor(output);

auto in = twml::getConstTensor(input);

twml::add1(\*out, \*in););

return TWML\_ERR\_NONE;

}

twml\_err twml\_copy(twml\_tensor output, const twml\_tensor input) {

HANDLE\_EXCEPTIONS(

auto out = twml::getTensor(output);

auto in = twml::getConstTensor(input);

twml::copy(\*out, \*in););

return TWML\_ERR\_NONE;

}

inline twml\_err twml\_get\_feature\_id\_internal(int64\_t \*result,

uint64\_t out\_size, uint16\_t \*out,

uint64\_t out2\_size, uint16\_t \*out2,

const uint64\_t len, const char \*str) {

uint64\_t k = 0;

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

if (str[i] == '#') {

k = i;

break;

}

}

uint8\_t hash[16];

if (k != 0) {

ssize\_t n = utf8\_to\_utf16((const uint8\_t \*) str, k, out, out\_size);

if (n < 0) throw std::invalid\_argument("error while converting from utf8 to utf16");

MurmurHash3\_x64\_128(out, n \* sizeof(uint16\_t), 0, out2);

n = utf8\_to\_utf16((const uint8\_t \*) (str + k + 1), len - k - 1, &out2[4], out2\_size - 8);

if (n < 0) throw std::invalid\_argument("error while converting from utf8 to utf16");

MurmurHash3\_x64\_128(out2, (n \* sizeof(uint16\_t)) + 8, 0, hash);

} else {

ssize\_t n = utf8\_to\_utf16((const uint8\_t \*)str, len, out, out\_size);

if (n < 0) throw std::invalid\_argument("error while converting from utf8 to utf16");

MurmurHash3\_x64\_128(out, n \* sizeof(uint16\_t), 0, hash);

}

int64\_t id;

memcpy(&id, hash, sizeof(int64\_t));

\*result = id;

return TWML\_ERR\_NONE;

}

static const int UTF16\_STR\_MAX\_SIZE = 1024;

twml\_err twml\_get\_feature\_id(int64\_t \*result, const uint64\_t len, const char \*str) {

try {

uint16\_t out[UTF16\_STR\_MAX\_SIZE];

uint16\_t out2[UTF16\_STR\_MAX\_SIZE];

return twml\_get\_feature\_id\_internal(result,

UTF16\_STR\_MAX\_SIZE, out,

UTF16\_STR\_MAX\_SIZE, out2,

len, str);

} catch(const std::invalid\_argument &ex) {

// If the space on the stack is not enough, try using the heap.

// len + 1 is needed because a null terminating character is added at the end.

std::vector<uint16\_t> out(len + 1);

std::vector<uint16\_t> out2(len + 1);

return twml\_get\_feature\_id\_internal(result,

len + 1, out.data(),

len + 1, out2.data(),

len, str);

}

}