Documentation Assignment 2

Martina Donadi, 865737; Dainese Fabio, 857661

December 23, 2019

1 Introduction

1.1 Assignment Requirements

In this assignment we had to expand the *first assignment* library of the tensors in such a way that will now offers also some operations between tensors using the Einstein's notation, like reported below:

- **Product** between one or more tensors;
- **Sum** between two or more tensors or product of tensors;
- **Trace** on one tensor.

By job specification the product of sums of tensors/products is not allowed.

The operations developed work with dynamically ranked tensors as well as with fixed-rank tensors by maintaining and checking as much static information as possible.

1.2 Einstein's Notation Definition

Einstein's notation was introduced in 1916 to simplify the notation used for general relativity, expresses tensor operations by rendering implicit the summation operations. According to this notation, repeated indices in a tensorial expression are implicitly summed over, so the expression:

$$a_{ijk} \cdot b_j$$

Represents a rank 2 tensor c indexed by i and k such that:

$$c_{ik} = \sum_j a_{ijk} \cdot b_j$$

This specific notation allows also for simple contractions, like the following:

$$Tr(a) = a_{ii}$$

As well as additions (subtractions) and multiplications.

2 Implementation

2.1 Tensors

The implementation consists of polymorphic classes as represented in the UML Class Diagram in 'Figure 2'. The *proxy_tensor* class represents a generic tensor and has two children classes *proxy_labeled_tensor* and *proxy_op_tensor*.
The *proxy_labeled_tensor* class represents a labeled tensor in the fact that it contains a list of labels associated to the dimensions of the tensor. This list allows one to fix a certain dimension and carry out tensor operations between two tensors according to the Einstein's notation. Considered the tensor $a_{i,j,k}$ and the tensor $b_{k,l,m}$, an operation carried out on the two is related to the third dimension of a and the first dimension of b. The important fact on this class is that there are defined the operators that generate the classes instances of the operations.
The sibling of *proxy_labeled_tensor* is *proxy_op_tensor*. This class aggregates the operations of sum and product defined as sub-classes. It also has the functions to do the evaluation of the operations which are made in a delayed operation and returning a tensor.
proxy_prod_tensor and *proxy_sum_tensor* respectively represent the classes for the product operation and sum operation. Their behaviour is quite similar to the *proxy_labeled_tensor* class except for the labels management.

A more clear explanation on how the tensors and their operations work is provided in the 'Section 3' where there will be analyzed some test cases.

2.2 Labels

The labels represents the named and ordered list of a tensor dimensions. Labels make it possible to apply operations to tensors according to the Einstein's notation. The implemented *label* class stores the *name* of the label, the *size* of the associated dimension and the vector of the *positions*, i.e. the sorted position numbers.

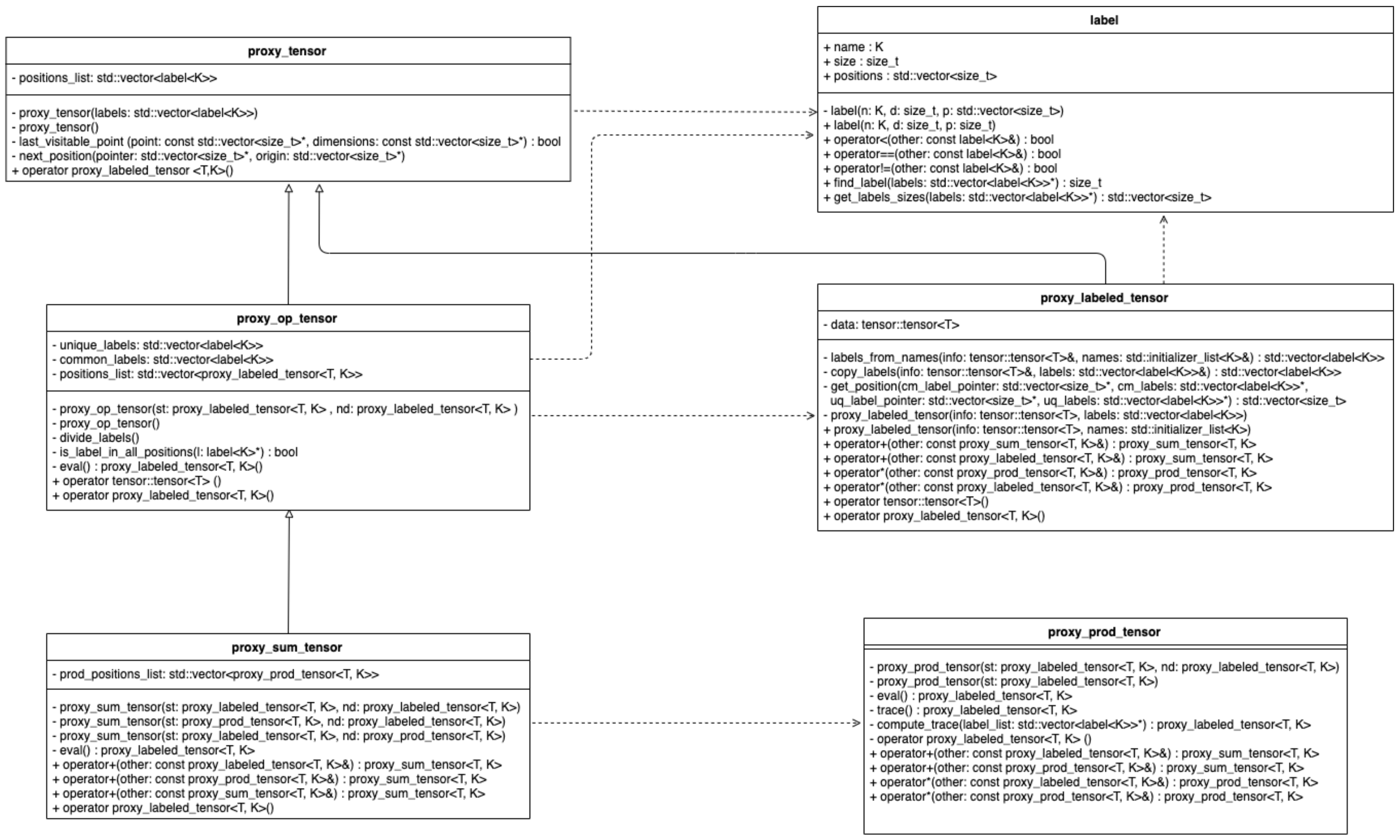
proxy_tensor
- positions_list: std::vector<label<K>>
- proxy_tensor(labels: std::vector<label<K>>)
- proxy_tensor()
- last_visitable_point (point: const std::vector<size_t>*, dimensions: const std::vector<size_t>*) : bool
- next_position(pointer: std::vector<size_t>*, origin: std::vector<size_t>*)
+ operator proxy_labeled_tensor <T,K>()
label
+ name : K
+ size : size_t
+ positions : std::vector<size_t>
- label(n: K, d: size_t, p: std::vector<size_t>)
+ label(n: K, d: size_t, p: size_t)
+ operator<(other: const label<K>&) : bool
+ operator==(other: const label<K>&) : bool
+ operator!=(other: const label<K>&) : bool
+ find_label(labels: std::vector<label<K>>*) : size_t
+ get_labels_sizes(labels: std::vector<label<K>>*) : std::vector<size_t>
proxy_op_tensor
- unique_labels: std::vector<label<K>>
- common_labels: std::vector<label<K>>
- positions_list: std::vector<proxy_labeled_tensor<T, K>>
- proxy_op_tensor(st: proxy_labeled_tensor<T, K> , nd: proxy_labeled_tensor<T, K>)
- proxy_op_tensor()
- divide_labels()
- is_label_in_all_positions(l: label<K>*) : bool
- eval() : proxy_labeled_tensor<T, K>()
+ operator tensor::tensor<T> ()
+ operator proxy_labeled_tensor<T, K>()
proxy_labeled_tensor
- data: tensor::tensor<T>
- labels_from_names(info: tensor::tensor<T>&, names: std::initializer_list<K>&) : std::vector<label<K>>
- copy_labels(info: tensor::tensor<T>&, labels: std::vector<label<K>>&) : std::vector<label<K>>
- get_position(cm_label_pointer: std::vector<size_t>*, cm_labels: std::vector<label<K>>*,
uq_label_pointer: std::vector<size_t>*, uq_labels: std::vector<label<K>>*) : std::vector<size_t>
- proxy_labeled_tensor(info: tensor::tensor<T>, labels: std::vector<label<K>>)
+ proxy_labeled_tensor(info: tensor::tensor<T>, names: std::initializer_list<K>)
+ operator+(other: const proxy_sum_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator+(other: const proxy_labeled_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator*(other: const proxy_prod_tensor<T, K>&) : proxy_prod_tensor<T, K>
+ operator*(other: const proxy_labeled_tensor<T, K>&) : proxy_prod_tensor<T, K>
+ operator tensor::tensor<T>()
+ operator proxy_labeled_tensor<T, K>()
proxy_sum_tensor
- prod_positions_list: std::vector<proxy_prod_tensor<T, K>>
- proxy_sum_tensor(st: proxy_labeled_tensor<T, K>, nd: proxy_labeled_tensor<T, K>)
- proxy_sum_tensor(st: proxy_prod_tensor<T, K>, nd: proxy_labeled_tensor<T, K>)
- proxy_sum_tensor(st: proxy_labeled_tensor<T, K>, nd: proxy_prod_tensor<T, K>)
- eval() : proxy_labeled_tensor<T, K>
+ operator+(other: const proxy_labeled_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator+(other: const proxy_prod_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator+(other: const proxy_sum_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator proxy_labeled_tensor<T, K>()
proxy_prod_tensor
- proxy_prod_tensor(st: proxy_labeled_tensor<T, K>, nd: proxy_labeled_tensor<T, K>)
- proxy_prod_tensor(st: proxy_labeled_tensor<T, K>)
- eval() : proxy_labeled_tensor<T, K>
- trace() : proxy_labeled_tensor<T, K>
- compute_trace(label_list: std::vector<label<K>>*) : proxy_labeled_tensor<T, K>
- operator proxy_labeled_tensor<T, K> ()
+ operator+(other: const proxy_labeled_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator+(other: const proxy_prod_tensor<T, K>&) : proxy_sum_tensor<T, K>
+ operator*(other: const proxy_labeled_tensor<T, K>&) : proxy_prod_tensor<T, K>
+ operator*(other: const proxy_prod_tensor<T, K>&) : proxy_prod_tensor<T, K>

Figure 1: UML Class Diagram.

3 Test Cases

In the *main* function of the '*tensor.cc*' file we've also included several simple test cases.

In the following sections we are going to analyze a couple of them.

3.1 Sum

In sum operation tensors have a one to one correspondence thus tensors sum is easier to compute than tensor product. Two tensors to be sum must have the same structure, i.e. they must have at least correspondent labels if not same ordered dimensions.

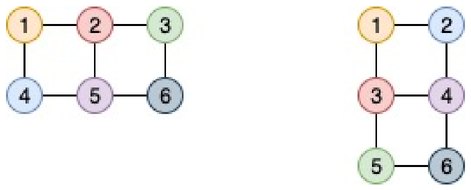

Figure 2: Tensor product.

3.2 Product

To keep the calculations easy two bi-dimensional tensors, i.e. with dimensions (n, m), have been multiplied. The first has size $(2, 3)$ and the second has size $(3, 2)$. In this case there is one label in common, (m), and the result tensor has the size of $(2, 2)$ labelled (n, p). For instance the first element of the result is computed below:

$$res_{0,0} = \sum_{m=0}^{2} a_{0,m} \cdot b_{m,0} = 22$$

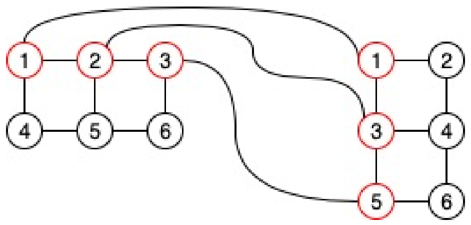

Figure 3: Tensor product.

3.3 Product with more than one common label

The product test with multiple labels is carried out on two tensors of dimensions: $a_{n=2,m=3,o=2}$ and $b_{m=3,n=2}$. In order to multiply them the product Einstein's notation formula is applied. Computing the first element holds:

$$res_0 = \sum_{n=0}^{1} \sum_{m=0}^{2} a_{n,m,0} \cdot b_{m,n} = 151$$

3.4 Trace

The implemented trace function has been designed in such a way that it can be use it not only with a single tensor, but also with product of tensors that haven't got unique labels.

So for example, if we consider the same tensors used in the '*Section 3.3*', but removing the '*o*' dimension to the first tensor, we will end up having a number as a result.

So if we want to apply the trace to a tensor ($n = 3$,$n = 3$). The result will be the sum of the diagonal as reported in the following image:

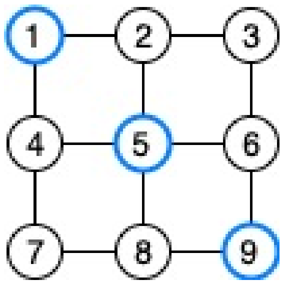

Figure 4: Tensor trace.