TENLab: Distributed Tensor-based Language

Xiangrong, Xu
xx2367@columbia.edu

Xincheng, Xie
xx2365@columbia.edu

Songqing, Ye
sy3006@columbia.edu

Senhong, Liu
sl4839@columbia.edu

Fall 2021

1 Overview

The TENLab programming language is an imperative, high-performance language for matrix computation. It is inspired by Python and Matlab, but can support distributed matrix computing. The basic syntax, e.g., function declaration, control flow etc., will be pretty much similar to the Python. However, the operators and matrix computation are inspired by the Matlab.

In contrast to Python and Matlab, variables in TENLab need to be declared statically, and TENLab provides three types of variables, namely int, float, and String. But it also allows a void type of tensor, which is a similar type of struct in C. They're all wrapped up in a tensor, and that's the underlying philosophy of our language, which is to say, **everything is a tensor**.

In the end, if possible, we would like to build a distributed model, with such a model, the matrix computation could be optimized. We could also try to support a user-defined distribution model to help accelerate matrix computation.

Our goals are:

- Implement a Python-style syntax language to support matrix computation.
- Potential optimization due to the static declaration of variable.
- If possible, implement a distributed models, e.g., MapReduce, to help accelerate the matrix computation.

2 Language Definition

2.1 Data Types and Operations

TENLab's primitive data types are 64-bit integers, 64-bit floats, and characters. Strings are a built-in a class wrapping immutable arrays of characters. This language will be statically typed and will offer many common operators for users to perform simple operations on tensors.

2.1.1 Basic Data Types

We support a total of three basic data types in tensors.

Type name	Description
int	64-bit signed integer
float	64-bit float-point number
char	8-bit character

Table 1: Basic Data Type

2.1.2 Arithmetic Operators

The Table 2 below shows basic arithmetic operations we plan to implement. Denote **A** and **B** as the first operand and the second operand in a binary operation. Constants is in the form of 0-dimension tensor, and it should only appear on the right side of operators. Operators with '.' means element-wise operations.

Operator Name	Functionality	Example
+	Addition(**A** and **B** with same dimension or **B** is 0-dim tensor)	$\mathbf{A} + \mathbf{B}$
-	Subtraction(**A** and **B** with same dimension or **B** is 0-dim tensor)	$\mathbf{A} - \mathbf{B}$
*	Multiplication (**A** should be an m-by-p tensor and **B** should be a p-by-n tensor)	$\mathbf{A} * \mathbf{B}$
.*	Element-wise multiplication (**A** and **B** with same dimension required)	$\mathbf{A}. * \mathbf{B}$
/	Division (**B** should be 0-dim tensor)	$\mathbf{A}/\mathbf{B}$
ˆ	Power (**B** should be 0-dim tensor)	$\mathbf{A}\hat{}\mathbf{B}$
.ˆ	Element-wise power (**A** and **B** with same dimension required)	$\mathbf{A}.\hat{}\mathbf{B}$
′	Transpose	$\mathbf{A}'$
%	Mod (**B** should be 0-dim tensor)	$\mathbf{A}\%\mathbf{B}$
//	Remainder (**B** should be 0-dim tensor)	$\mathbf{A}//\mathbf{B}$

Table 2: Arithmetic Operators

2.1.3 Relational Operators

Our language also offer relational operations. All the following relational operations will return a logical tensor with elements set to logical 1 (**true**) where tensors **A** and **B** satisfies the operation; otherwise, the element is logical 0 (**false**).

Operator name	Description
==	Determine equality
>=	Determine greater than or equal to
>	Determine greater than
<=	Determine less than or equal to
<	Determine less than
! =	Determine inequality

Table 3: Relational Operators

2.1.4 Logical Operators

TENLab also provides logical AND, OR and NOT operations. Logical operators should only appear between two expressions.

Operator name	Description
&&	Logical OR operator
||	Logical AND operation
!	Logical NOT than

Table 4: Logical Operators

2.2 Keywords

We want to make the number of keywords as few as possible. The following keywords are included in TENLab:

Control: *if, elif, else, for, while, in, continue, break, return, read, print, exit*

Type: *int, float, string, void*

Tensor related: *cat* (concatenate two tensors), *shape* (return the shape of the tensor)

Common tensor operations will be supported in the standard library.

2.3 Built-in Functions

Following are some examples of built-in function samples:

any(x): takes one int or float tensor as argument, returns 0 if all the elements are zero; otherwise, returns 1

all(x): takes one int or float tensor as argument, returns 0 if any of the element is zero; otherwise, returns 1

sum(x): takes one int or float tensor as argument, returns the sum of all elements in a 0-d tensor

zeros(x): takes one int tensor as argument, returns an int tensor of that shape which is filled by zeros

ones(x): takes one int tensor as argument, returns an int tensor of that shape which is filled by ones

And there will be more...

2.4 Control Flow

We do not have Bool type in our language. Thus, in control flows, we take one constant (0 dim tensor) as the condition. The constant being zero means false; otherwise, it is true. By default, we do not accept tensors of other shapes as condition.

2.4.1 if

An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 value.

The else statement is an optional statement and there could be at most only one else statement following if.

```
if (condition) {
    # statement
}
elif (condition) {
    # statement
}
else {
    # statement
}
```

2.4.2 while

Repeats a statement or group of statements while a given variable is not zero. It tests the condition before executing the loop body.

```
while (condition) {
    # statement
}
```

2.4.3 for

Executes a sequence of statements multiple times. **For** takes an int variable and an 1-d tensor as argument. The int variable will traverse the 1-d tensor during the loop.

```
# do not need to define i in advance
for (i in [1:10:1]) {
    # statement
}
```

2.5 Comments

```
int x = 0 # this is a comment
''' This is
a multi-line
comment
'''
```

2.6 Functions

In our language, a function can take multiple arguments and return a tensor which can be void-type to allow the return of multiple values.

```
def foo (a, b) {
    # statement
    return cat(c, d)
}
```

3 Sample

```
# Markov Process
P = float([[3/4, 1/4], [1/4, 3/4]]) # transition matrix
s = float([[0.2, 0.8]]) # initial state
# judge difference of two matrices' elements is less than 1e-5
def diff(prev, curr)
{
    if (shape(prev) != shape(crr))
    {
        exit(-1)
    }
    delta = prev - cur
    flag = 1
    shape_t = shape(delta)
    for (i in 0:shape_t[0]:1)
    {
        for (d in delta[i,0:shape_t[1]:1])
        {
            if (abs(d) > 1e-5)
```

```
            {
                flag = 0
            }
        }
    }
    return flag
}
# multiply state and transition matrix
def mulPs(s, P)
{
    return s, s * P
}
# check state after four transitions
print(s * (P^4))
# Iterates until stable
s_prev = zeros(shape(s))
while (diff(s_prev, s) == 0)
{
    s_prev, s = mulPs(s, P)
}
```

4 Reference

[1] Digo: distributed golang

http://www.cs.columbia.edu/ sedwards/classes/2021/4115-spring/proposals/Digo.pdf

[2] Facelab: A Facial Image Editing Language

http://www.cs.columbia.edu/ sedwards/classes/2017/4115-fall/proposals/Facelab.pdf