**UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG**
SCHOOL OF ELECTRICAL AND INFORMATION ENGINEERING

# ELEN4020A: Data Intensive Computing: Lab 1

Sbonelo Mdluli(1101772), Heemal Ryan(792656), Haroon Rehman(1438756)

February 25, 2019

## 1 INTRODUCTION

Loosely defined a tensor is a mathematical object that generalizes the notion of vectors to higher dimensions. Tensors provide a natural way of representing multidimensional data. Tensors have many applications in physics, engineering and machine learning. An $n^{th}$ rank tensor in $m$ dimensional space has $m^n$ components. Tensors consist of multiple operations such as addition and multiplication. This lab report discusses code written in Python 3.6 that is used to perform 2-D and 3-D tensor addition and multiplication.

## 2 PROCEDURES

Two matrices of dimensions $n \times n$ (or $n \times n \times n$ for 3-D) with range-based randomly generated values are created. The number of columns in the first matrix are checked to be the same number of rows in the second matrix. However, for this specific task, we have set it that only square and cubic matrices be generated as instructed. The output from the operations is formatted such that the first $n$ rows and columns represent one matrix, where $n$ is the dimension of the tensor.

### 2.1 2-D matrix addition

Tensor addition of tensor A and tensor B is referred to as the addition corresponding elements to result a tensor of same dimensions as A and B. In this case n=2, so therefore the addition of each corresponding elements in the two $2 \times 2$ matrices gives the resultant $2 \times 2$ matrix. The use of for loops allows for the correct addition to occur. The procedure is detailed by algorithm 1. The rank2TensorAdd algorithm has a worst-case running time of $\mathcal{O}(n^2)$.

**Algorithm 1:** Rank 2 Tensor Addition for Summing 2-D by 2-D Tensors

```
Function name: rank2TensorAdd
Initialization :
A an B are n x n matrices
Let C be an empty n x n matrix
for i= 1 to n do
    for j=1 to n do
        C[i][j] = A[i][j] + B[i][j]
    end
end
return C
```

## 2.2 2-D matrix multiplication

In this lab, tensors are restricted to $n$ dimensional square tensors which simplifies the multiplication computation. An empty $2 \times 2$ matrix of size equal to the one of the matrices being added is created. The resultant values are stored in the empty matrix. Equation 2.1 describes tensor multiplication of $2 \times 2$ matrices and how each element is generated in the resultant matrix.

$$C_{ij} = \sum_{k}^{n} A_{ik} \times B_{kj} \tag{2.1}$$

Algorithm 2 provides the pseudo-code for the $2 \times 2$ matrix multiplication. The rank2TensorMult algorithm has a worst-case running time of $\mathcal{O}(n^3)$.

**Algorithm 2:** Rank 2 Tensor Multiplication for computing 2-D by 2-D Tensor Product

```
Function name: rank2TensorMult
Initialization :
A an B are n x n matrices
Let C be an empty n x n matrix
for i= 1 to n do
    for j=1 to n do
        C[i][j] = 0
        for k= 1 to n do
            C[i][j] = C[i][j] + A[i][k] × B[k][j]
        end
    end
end
return C
```

## 2.3 3-D matrix addition

Individual layers and added together to give a layer of same dimensions and similar to the 2-D tensor addition, each of the corresponding elements in both layers are added to obtain the resultant layer. The layers are concatenated together to give the resultant 3-D matrix .Again the use of a for loop in Python allows for this type of usage. Algorithm 3 contains pseudo-code for this operation. This algorithm has a worst-case running time of $\mathcal{O}(n^3)$.

**Algorithm 3:** Rank 3 Tensor Addition for summing 3-D by 3-D Tensors

```
Function name: rank3TensorAdd
Initialization :
A an B are n x n x n matrices
Let C be an empty n x n x n matrix
for i= 1 to n do
    C[i] = rank2TensorAdd(A[i],B[i])
end
return C
```

## 2.4 3-D matrix multiplication

Each 3-D matrix is processed in layers of 2-D matrices. To perform the multiplication, we split matrix A into horizontal planes and matrix B into vertical ones. Each horizontal plane in A multiplies every vertical plane in B. The horizontal plane is split into rows and the vertical plane into columns. Each row in the horizontal plane multiplies every column in the vertical plane, this is done using the rank2TensorMult procedure. Further processing in done on the row and columns before being passed into the rank2TensorMult procedure. The row and column vectors are converted to $n \times n$ matrices by appending zeros in the remaining indices. This conversion is necessary since the rank2TensorMult only takes square matrices. Normally the multiplication of vectors result in a single value however since we use the rank2TensorMult procedure to compute the row and column multiplication, the result we get from the procedure is stored in a square matrix. Algorithm 4 below contains pseudo-code for the 3-D matrix multiplication.

**Algorithm 4:** Rank 3 Tensor Multiplication for computing 3-D by 3-D Tensor Product

```
Function name: rank3TensorMult
Initialization :
A an B are n x n x n matrices
Let C be an empty n x n x n matrix
Let hplane = a = b be empty n x n matrices
tmp = rslt = 0
for i= 1 to n do
    hplane = A[i]
    for j= 1 to n do
        vplane = extract_column(B,j)
        for k= 1 to n do
            for x= 1 to n do
                VplaneColumn = extract_column(vplane,x)
                matA= RowvecToMat(hplane(k))
                matB= ColvecToMat(VplaneColumn)
                tmp= rank2TensorMult(a,b)
                rslt = tmp[0][0]
                C[x][i][k]+=rslt
            end
        end
    end
end
return C
```

## 3 Conclusion

The algorithms were tested for correctness using $n = 2$. One can further verify the results using numpy for the rank2Tensor procedures. From the algorithm analysis it is evident that computationally complexity increases with the rank and dimensions of the tensor. The procedures developed in this lab are presented for $n \times n$ or $n \times n \times n$ however modifications may be made to make them more flexible to work in varying dimensions.