Matrix Processing

Miguel Castillo II & Aleksander Jaworowski


KEANU REEVES
LAURENCE FISHBURNE
MATRIX
ON APRIL 2ND THE FIGHT FOR THE FUTURE BEGINS

$$\begin{pmatrix} 2 & 5 & 7 & 8 \\ 1 & 2 & 3 & 1 \\ 4 & 5 & 0 & 1 \end{pmatrix}$$

What is a Matrix?

Simply put:

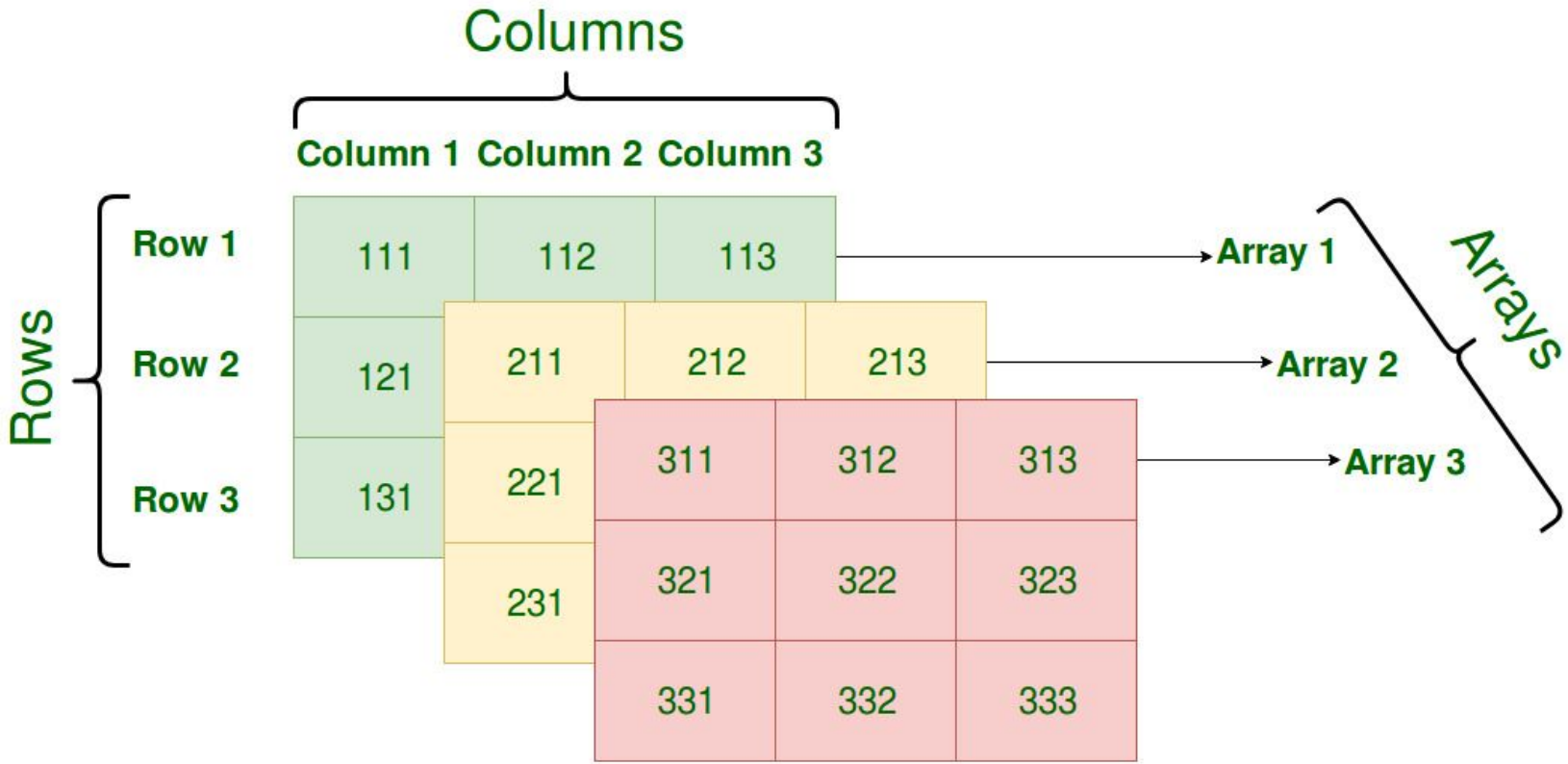
A Matrix is an arrangement of numbers into rows and columns or an array of numbers

A table is an example of a matrix

This table can be defined as having 2 dimensions

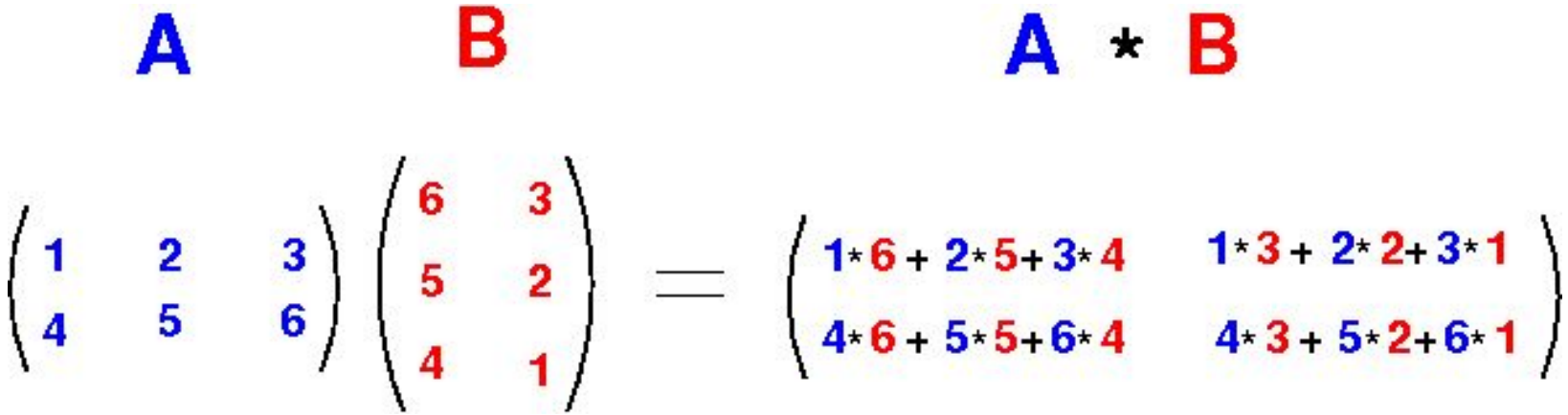
		Category 1		
		Group 1	Group 2	Group 3
	Group 1	a	d	g
Category 2	Group 2	b	e	h
	Group 3	c	f	i

where ***a***, ***b***, ***c***, ***d***, ***e***, ***f***, ***g***, ***h***, and ***i*** are the number of observations in each cell.

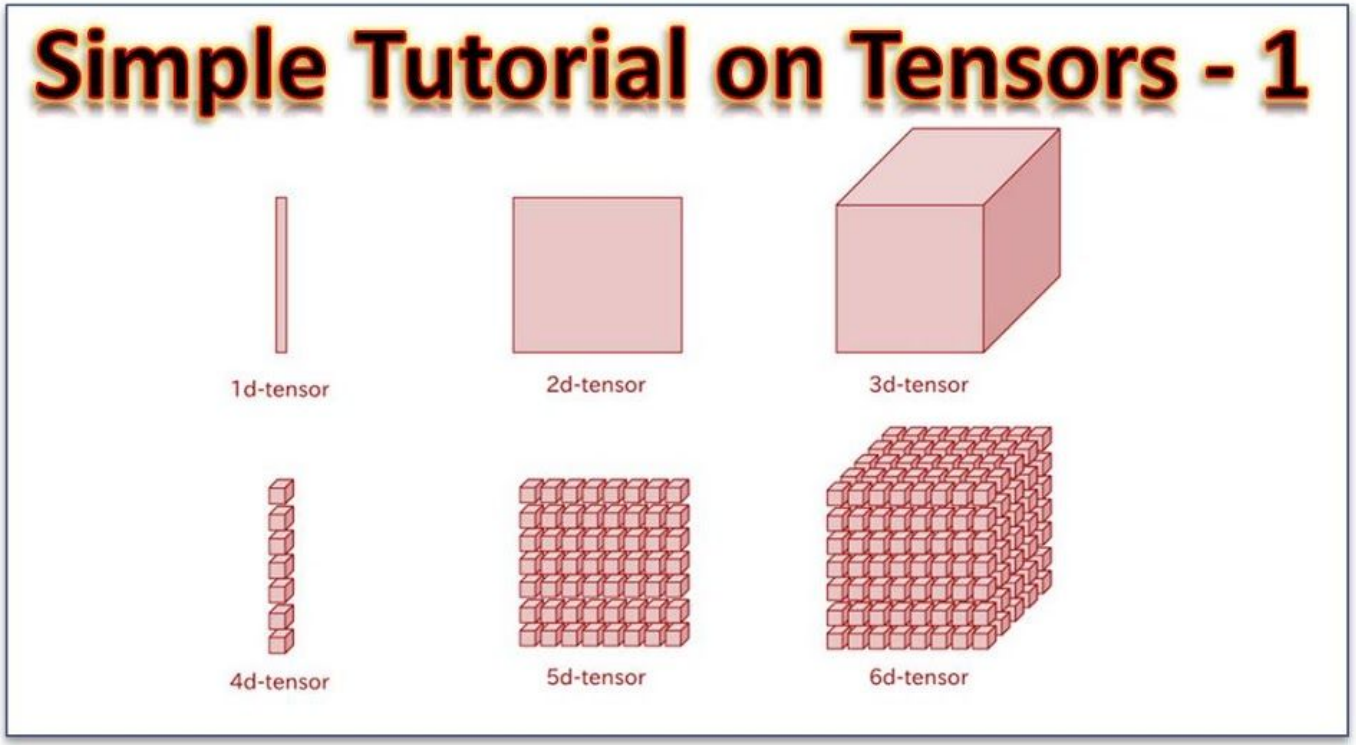

Columns
Column 1
Column 2
Column 3
Rows
Row 1
Row 2
Row 3
111
112
113
121
211
212
213
131
221
311
312
313
231
321
322
323
331
332
333
Array 1
Array 2
Array 3
Arrays

Index	0	1	2	3	4
0	65,340	12,483	138,189	902,960	633,877
1	5,246	424,642	650,380	821,254	866,122
2	89,678	236,781	601,691	329,274	913,534
3	103,902	1,567	733,611	263,010	85,550
4	2,778	658,305	128,788	978,155	620,702
5	45,024	55,058	705,586	89,672	384,605
6	780	47,538	523,784	556,801	617,107
7	32,667	350,890	834,753	638,108	85,188
8	56,083	145,582	775,040	548,322	756,587
9	41,123	543,542	537,738	513,048	418,482

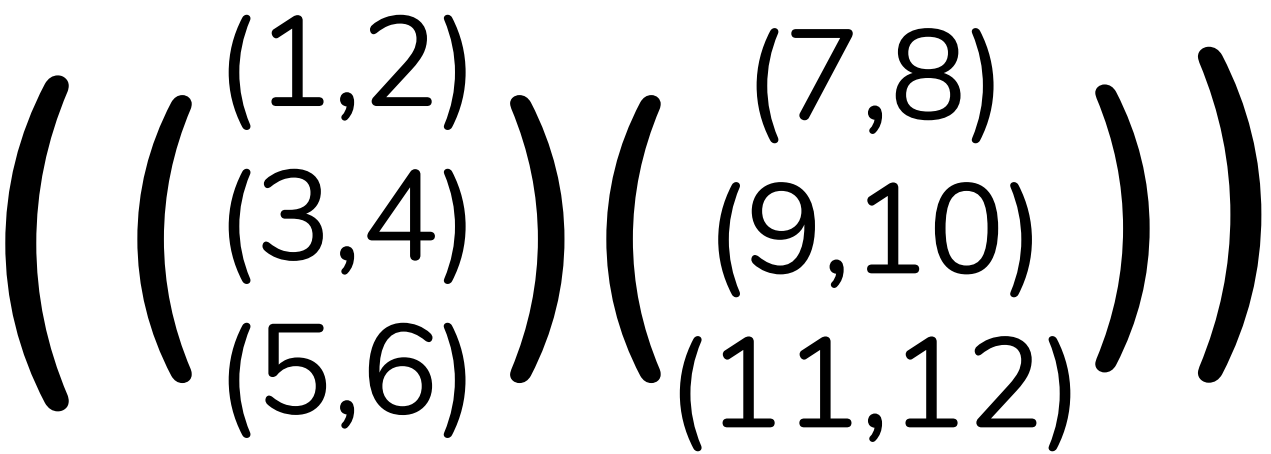
Matrix Manipulation



Tensors


Simple Tutorial on Tensors - 1
1d-tensor
2d-tensor
3d-tensor
4d-tensor
5d-tensor
6d-tensor

Scala Tensor with parameters: 2, 3, 2



DESIGN

Thus far

- New class representing multidimensional structures
- Expanding environment with functions to create and act on multidimensional structures and our old beloved vector
- Might instead move to only having VVector, with dimensions as 1 per default

Implementation

```
case class VTensor (val l: List[Value], val ndim: List[Value] ) extends Value
```

```
var allArr: Array[Iterable[_]] = (0 until ndim.length).map(x => List(x)).toArray
def thisTensor = {
  allArr(0) = l.grouped(ndim.last.getInt).toList
  var n = 1
  for ( i <- ndim.reverse.tail) {
    allArr(n) = allArr(n-1).grouped(i.getInt).toList
    n += 1
  }
  allArr.last
}
```

Implementation: shell()

```
def operFold ( vs : List[Value] ) : Value = {
  if ( vs(0).isVector ) {
    val l = vs(0).getList
    val dims = vs.tail
    val result = VTensor(l,dims)
    return result
  }
  else return new VString("first element must be of type VECTOR")
}
```

Implementation: helpers

```
def operFill (vs : List[Value]) : Value = {
  if (vs(0).isInteger && vs(0).isInteger)  {
    val offLength = vs(0).getInt
    val fillWith = vs(1).getInt
    val myVec = List.fill(offLength)(fillWith)
    val result = new VVector(myVec.map(x => ci(x)))
    return result
  } else  println( vs) ; return new VBoolean(false)
}

def operRange (vs : List[Value]) : Value = {
  val myFrom = vs(0).getInt
  val myTo = vs(1).getInt
  var result: Value = new VVector(List(ci(0)))
  if ( vs.length == 2) {
    result = new VVector(myFrom.to(myTo).toList.map(x => ci(x)))
  } else if( vs.length == 3){
    val myBy = vs(2).getInt
    result = new VVector(myFrom.to(myTo).by(myBy).toList.map(x => ci(x)))
  } else println(" wrong number arguments ")
  return result
}
```

Implementation: multiplication

```
def operTimes (vs : List[Value]) : Value = {  // APL like pointwise multiplication
  checkNumberArgs(vs, 2)
  val v1 = vs(0)
  val v2 = vs(1)
  if ( v1.isFraction && v2.isFraction) { // pure multiplication
    val n1 = v1.getNumerator()
    val n2 = v2.getNumerator()
    val d1 = v1.getDenominator()
    val d2 = v2.getDenominator()
    return new VFraction(n1 * n2, d1 * d2).simplify()
  } else if (v1.isInteger && v2.isVector ) {  // Integer times vector
      val result = v2.getList.map( x => ci(x.getInt * v1.getInt) )
      return new VVector(result)
    } else if ( v1.isInteger && v2.isTensor ) { // Integer times tensor
      val listRes = v2.getList.map( x => ci(x.getInt * v1.getInt) )
      val dims = v2.getDim
      val result = new VTensor(listRes, dims)
      return result
    } else if ( v1.isVector && v2.isVector ) { // Vector times Vector
      val l1 = v1.getList
      val l2 = v2.getList
      require( l1.length == l2.length, "not of same length") // expand smaller
      var result = Array.fill(l1.length)(0)
      for ( i <-  l1.indices) {
        result(i) = l1(i).getInt * l2(i).getInt
      }
      return new VVector( result.map(x => ci(x)).toList )
    }  else if ( vs(0).isTensor && vs(1).isTensor ) { // tensor times tensor of same dim, should be like vector
        if (vs(0).getDim == vs(1).getDim) {
          val l1 = vs(0).getList
          val l2 = vs(1).getList
          var addArr: Array[Int] = Array()
          for( i <- l1.indices) { addArr = addArr :+ ( l1(i).getInt * l2(i).getInt ) }
          val result = VTensor( addArr.map(x => ci(x)).toList, vs(0).getDim )
          return  result
        }
```

Challenges

Understanding sources talking about APL

Multiple choices, hard to oversee effect in beginning

Arbitrary dimensions generally makes ‘ol reliable for loop a bad alternative

Math

Matrix requirements

Understanding n-dimension data with text

Demo

(fill 10 1)

#def l1 (range 2 16 2)

#def t1 (fold l1 2 2 2)

#def l2 (range 1 16)

#def t2 (fold l2 2 2 2 2)

#def t3 (fold l2 2 2 4)

(+ t1 t1)

(* t1 t1)

(square t1)