Machine Learning(ML): Assignment 2

Charvi Dave J015

Task1: Prove multiplication properties of matrices

In [1]:

```
import numpy as np
P= np.array([[5,1,1],[7,8,9],[75,5,89]])
Q= np.array([[6,8,12],[13,12,11],[8,9,10]])
R= np.array([[5,1,21],[65,66,67],[68,69,70]])
I= np.identity(3)
```

In [2]:

```
print('Matrix P: \n',P)
print('Matrix Q: \n',Q)
print('Matrix R: \n',R)
print('Identity Martix: \n',I)
```

```
Matrix P:
 [[ 5  1  1]
 [ 7  8  9]
 [75  5 89]]
Matrix Q:
 [[ 6  8 12]
 [13 12 11]
 [ 8  9 10]]
Matrix R:
 [[ 5  1 21]
 [65 66 67]
 [68 69 70]]
Identity Martix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Commutative Property (does not hold true):

In [3]:

```
P_dot_Q = P.dot(Q)
Q_dot_P = Q.dot(P)
```

In [4]:

```
print('P.Q: \n',P_dot_Q)
print('Q.P: \n',Q_dot_P)
```

```
P.Q:
 [[  51   61   81]
 [ 218  233  262]
 [1227 1461 1845]]
Q.P:
 [[ 986  130 1146]
 [ 974  164 1100]
 [ 853  130  979]]
```

Associative property:

In [5]:

```
PQ_R = np.dot(P, Q).dot(R)
P_QR = P.dot(np.dot(Q, R))
```

In [6]:

```
print('(P.Q).R: \n',PQ_R)
print('(P.(Q.R): \n)',P_QR)
```

```
(P.Q).R:
 [[  9728   9666  10828]
 [ 34051  33674  38529]
 [226560 224958 252804]]
(P.(Q.R):
) [[  9728   9666  10828]
 [ 34051  33674  38529]
 [226560 224958 252804]]
```

Distributive property:

In [7]:

```
lhs = np.dot(P, Q+R)
rhs = np.dot(P, Q) + np.dot(P, R)
```

In [8]:

```
print('P.(Q+R): \n',lhs)
print('P.Q + P.R: \n',rhs)
```

```
P.(Q+R):
 [[ 209  201  323]
 [1385 1389 1575]
 [7979 8007 9985]]
P.Q + P.R:
 [[ 209  201  323]
 [1385 1389 1575]
 [7979 8007 9985]]
```

Identity property:

In [9]:

```
PI = np.dot(P, I)
IP = np.dot(I, P)
```

In [10]:

```
print('P.I: \n',PI)
print('I.P: \n',IP)
```

```
P.I:
 [[ 5.  1.  1.]
 [ 7.  8.  9.]
 [75.  5. 89.]]
I.P:
 [[ 5.  1.  1.]
 [ 7.  8.  9.]
 [75.  5. 89.]]
```

Multiplicative property of zeros:

In [11]:

```
a = np.zeros(9).reshape(3, 3)
lhs = np.dot(P, a)
rhs = np.dot(a, P)
```

In [12]:

```
print('P.0: \n', lhs)
print('0.P: \n',rhs)
```

```
P.0:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
0.P:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Dimensions on matrix multiplication:

In [13]:

```
a,b,c = 7,8,9
mat_a_b = np.random.randn(a, b)
mat_b_c = np.random.randn(b, c)
mat_multi = np.dot(mat_a_b, mat_b_c)
result_x, result_y = mat_multi.shape
```

In [14]:

```
print(f'{a}x{b} matrix X {b}x{c} matrix = {result_x}x{result_y} matrix')
```

```
7x8 matrix X 8x9 matrix = 7x9 matrix
```

Task2: Inverse of a matrix:

In [15]:

```
Q_inv = np.linalg.inv(Q)
Q_inv
```

Out[15]:

```
array([[ 0.5       ,  0.66666667, -1.33333333],
       [-1.        , -0.85714286,  2.14285714],
       [ 0.5       ,  0.23809524, -0.76190476]])
```

Task3: Comparison of time between numpy and loops:

In [16]:

```
import time
size = 2000
numpy_mat_A = np.random.randn(size, size)
numpy_mat_B = np.random.randn(size, size)
list_mat_A = [list(i) for i in numpy_mat_A]
list_mat_B = [list(i) for i in numpy_mat_B]
```

In [17]:

```
loop_st = time.time()
list_mat_C = []
for i in range(size):
    row = []
    for j in range(size):
        row.append(list_mat_A[i][j] + list_mat_B[i][j])
    list_mat_C.append(row)
loop_ed = time.time()
```

In [18]:

```
numpy_st = time.time()
numpy_mat_C = numpy_mat_A + numpy_mat_B
```

```
numpy_ed = time.time()
```

In [19]:

```
print('Loops: ', loop_ed - loop_st)
print('Numpy: ', numpy_ed - numpy_st)
```

```
Loops:  1.3093974590301514
Numpy:  0.02393627166748047
```

Conclusion: Numpy is way much faster than the normal loops.