Machine Learning Assigment - 2

-Ahnaan Merchant j036

Prove Properties of Matrix Multiplication

In [9]:

```
import numpy as np
import time

a1=[[1,2,3],[4,5,6],[7,8,9]]
b1=[[10,11,12],[13,14,15],[16,17,18]]
c1=[[19,20,21],[22,23,24],[25,26,27]]

mat_1=np.array(a1)
mat_2=np.array(b1)
mat_3=np.array(c1)

print('Matrix A:\n')
print(mat_1)
print('\nMatrix B:\n')
print(mat_2)
print('\nMatrix C:\n')
print(mat_2)
```

```
Matrix A:

[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix B:

[[10 11 12]
 [13 14 15]
 [16 17 18]]

Matrix C:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

1)Non-Commutative

In [11]:

```
AB=np.dot(mat_1,mat_2)
BA=np.dot(mat_2,mat_1)
print('Dot Product A.B:\n')
print(AB)
print('\nDot Product B.A:\n')
print(BA)
```

```
Dot Product A.B:

[[ 84  90  96]
 [201 216 231]
 [318 342 366]]

Dot Product B.A:

[[138 171 204]
 [174 216 258]
 [210 261 312]]
```

In [13]:

```
#dot product isnt same so this is non commutative
#only commutative if matrix is an identity matrix
#A.I=I.A
```

In [16]:

```
mat_4=np.identity(3,dtype=int)
AI=np.dot(mat_1,mat_4)
IA=np.dot(mat_4,mat_1)

print('Dot Product A.I:\n')
print(AI)
print('\nDot Product I.A:\n')
print(IA)
```

```
Dot Product A.I:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Dot Product I.A:

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

2)Distributive

In [19]:

```
#A.(B+C)
mat_5=mat_2+mat_3
mat_6=np.dot(mat_1,mat_5)

#A.B+A.C
mat_7=np.dot(mat_1,mat_2)
mat_8=np.dot(mat_1,mat_3)
mat_9=mat_7+mat_8

print('A.(B+C)\n')
print(mat_6)

print('\nA.B+A.C\n')
print(mat_9)
```

```
A.(B+C)

[[222 234 246]
 [537 567 597]
 [852 900 948]]

A.B+A.C

[[222 234 246]
 [537 567 597]
 [852 900 948]]
```

3)Associative

In [22]:

```
#A.(B.C)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
mat_10=np.dot(mat_1,np.dot(mat_2,mat_3))
#(A.B).C
mat_11=np.dot(np.dot(mat_1,mat_2),mat_3)

print('A.(B.C)\n\n',mat_10)
print('\n(A.B).C\n\n',mat_11)
```

```
A.(B.C)

 [[ 5976  6246  6516]
 [14346 14994 15642]
 [22716 23742 24768]]

(A.B).C

 [[ 5976  6246  6516]
 [14346 14994 15642]
 [22716 23742 24768]]
```

Calculating inverse of a Matrix

In [32]:

```
mat_12=np.random.randint(100,size=(3,3))
mat_12
```

Out[32]:

```
array([[44, 94, 45],
       [33, 33, 21],
       [63, 72, 42]])
```

In [33]:

```
mat_12_inv=np.linalg.inv(mat_12)
mat_12_inv
```

Out[33]:

```
array([[-0.06635071, -0.3728278 ,  0.25750395],
       [-0.03317536, -0.51974724,  0.29541864],
       [ 0.1563981 ,  1.45023697, -0.86887836]])
```

Speed Difference Between Numpy and Traditional Looping

In [36]:

```
mat_13=np.random.randint(500,size=(10000,10000))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
mat_14=np.empty((10000,10000))

print('Dimensions Of Both The Matrices:')
print('Matrix A:',mat_13.shape)
print('Matrix B:',mat_14.shape)
```

```
Dimensions Of Both The Matrices:
Matrix A: (10000, 10000)
Matrix B: (10000, 10000)
```

In [37]:

```
#Looping
start=time.time()
for _ in range(len(mat_13)):
    for __ in range(len(mat_13)):
        mat_13[_][__]=mat_13[_][__]+13

end=time.time()
print('Time taken:',end-start)
```

```
Time taken: 67.84772539138794
```

In [38]:

```
#numpy
start=time.time()
mat_14=mat_13+13
end=time.time()

print('Time Taken:',end-start)
```

```
Time Taken: 0.15504908561706543
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD