Data Science Lab

Vasudevan T V

Matrix Operations

- Using Vectorisation
- Here various matrix operations are performed without using loops
- For this, we can use various functions in the built in package numpy

```
# Matrix Addition
>>> import numpy
>>> matrix1=numpy.matrix([[1,2],[3,4]])
>>> matrix2=numpy.matrix([[4,3],[2,1]])
>>> matrix3=numpy.add(matrix1,matrix2)
>>> print(matrix3)
[[5 5]
 [5 5]]
```

Matrix Operations

```
# Matrix Subtraction
>>> import numpy
>>> matrix1=numpy.matrix([[2,2],[2,2]])
>>> matrix2=numpy.matrix([[1,1],[1,1]])
>>> matrix3=numpy.subtract(matrix1,matrix2)
>>> print(matrix3)
[[1 1]
 [1 1]]
# Matrix Multiplication
>>> import numpy
>>> matrix1=numpy.matrix([[2,2],[2,2]])
>>> matrix2=numpy.matrix([[1,1],[1,1]])
>>> matrix3=numpy.matmul(matrix1,matrix2)
>>> print(matrix3)
[[4 4]
 [4 4]]
```

Matrix Operations

```
# Scalar Multiplication
>>> import numpy
>>> matrix1=numpy.matrix([[2,2],[2,2]])
>>> matrix2=2*matrix1
>>> print(matrix2)
[[4 4]
 [4 4]]
# Matrix Transpose
>>> import numpy
>>> matrix1=numpy.matrix([[1,2],[3,4]])
>>> print(matrix1)
[[1 2]
 [3 4]]
>>> matrix2=numpy.transpose(matrix1)
>>> print(matrix2)
[[1 3]
 [2 4]]
```

Matrix Transformations

- We can use matrices for performing various geometric transformations such as translation, rotation, scaling etc.
- Translation is the process of moving an object to a different position
- Rotation is the process of changing the angle of the object
- Scaling is the process of changing the size of objects

Matrix Transformations

- Translation Matrix

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Program

```
import numpy
def translationMatrix(tx=0, ty=0):
    return numpy.matrix([[1,0,tx],
                         [0,1,ty],
                         [0,0, 1]])
matrix=translationMatrix(1,1)
print(matrix)
```

Matrix Transformations

- Rotation Matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Program

```
import numpy
def rotationMatrix(degree):
    theta = numpy.radians(degree)
    c,s=numpy.cos(theta),numpy.sin(theta)
    return numpy.matrix([[c, -s, 0],
                         [s,  c, 0]
                         [0,  0, 1]])
matrix=rotationMatrix(30)
print(matrix)
```

Matrix Transformations

- Scaling Matrix

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Program

```
import numpy
def scalingMatrix(sx=0, sy=0):
    return numpy.matrix([[sx,0,0],
                         [0,sy,0],
                         [0, 0,1]])
matrix=scalingMatrix(2,2)
print(matrix)
```

Singular Value Decomposition (SVD)

- It is the process of decomposing a matrix into 3 components which are also matrices
- A matrix M is decomposed into 3 matrices U, S and V
- If M is a real matrix, U and V are orthogonal matrices and S is a diagonal matrix
- The advantage of such a decomposition is that we can do the subsequent matrix operations faster
- Applications - solving homogeneous linear equations, pattern recognition, natural language processing, weather prediction, machine learning etc.

Singular Value Decomposition (SVD)

- Program

```
# Imports matrix, matmul and diag functions only
from numpy import matrix
from numpy import matmul
from numpy import diag
# Imports svd fn from linalg(linear algebra) submodule of
# scipy module
from scipy.linalg import svd
# define a matrix
A = matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# Singular-value decomposition
# A is decomposed into 3 matrices U, a diagonal matrix
# and V
# Here S contains only the diagonal elements of the
# diagonal matrix
U, S, V = svd(A)
```

Singular Value Decomposition (SVD)

- Program - continued

```
print(U)
print(S)
print(V)
# create diagonal matrix from diagonal elements
Sigma = diag(S)
print(Sigma)
# reconstruct matrix
B = matmul(U,matmul(Sigma,V))
print(B)
```

Singular Value Decomposition (SVD)

- Output

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[-0.21483724  0.88723069  0.40824829]
 [-0.52058739  0.24964395 -0.81649658]
 [-0.82633754 -0.38794278  0.40824829]]
[1.68481034e+01 1.06836951e+00 4.41842475e-16]
[[-0.47967118 -0.57236779 -0.66506441]
 [-0.77669099 -0.07568647  0.62531805]
 [-0.40824829  0.81649658 -0.40824829]]
[[1.68481034e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.06836951e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 4.41842475e-16]]
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

Histogram

- Write a program to plot a histogram of marks obtained by students in a class
- Marks - 22,87,5,43,56,73,55,54,11,20,51,5,79,31,27

```
# imports pyplot, a module used in the package matplotlib
# to plot various figures
from matplotlib import pyplot
# imports array() from numpy package
from numpy import array
# subplots() specify the number of plots in the figure
# first argument is number of rows
# second argument is number of columns
# This function returns a tuple containing figure and axes
# objects
# These objects are assigned to fig and ax
# They are needed for changing figure level and axes level
# attributes
fig,ax = pyplot.subplots(1,1)
```

Histogram

- Program

```
a = array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
# Draws a histogram, first argument is the array of
# numbers, second argument bins are intervals of values
ax.hist(a,bins=[0, 10, 20, 30, 40, 50, 60, 70, 80,90,100])
ax.set_title("histogram of result")
ax.set_xticks([0, 10, 20, 30, 40, 50, 60, 70, 80, 90,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
# Shows the plot
pyplot.show()
```

Histogram

- Output