numpy-advance

June 10, 2023

1 Advance Numpy

1.1 Numpy array vs Python lists

[8]:
```
# speed
# list
a= [i for i in range (10000000)]
b= [i for i in range (10000000, 20000000)]
c=[]

import time
start= time.time()
for i in range(len(a)):
    c.append(a[i]+b[i])
print(time.time()-start)
```

```
4.620081186294556
```

[9]:
```
import numpy as np
a= np.arange(10000000)
b= np.arange(10000000, 20000000)

start= time.time()
c= a+b
print(time.time()-start)
```

```
0.18453383445739746
```

[10]:
```
4.62/0.18
```

[10]:
```
25.666666666666668
```

[11]:
```
# numpy uses c type array, it is a static array and it is not a referential␣
 ↪array
# (size remains fixed doesn't increase or decrese) item stores in memory not in␣
 ↪address
# list is a dynamic array which means everytime size doubles and it is a␣
 ↪referential array which means
```

```
# item is not stored directly, item is stored into specific address which takes additional time
# so numpy is very fast than list
```

[12]:

```
# memory
a = [i for i in range(10000000)]
import sys

sys.getsizeof(a)
```

[12]: 89095160

[13]:

```
# in numpy we can change the datatype as per our requirement
# list is taking more memory in bytes than numpy

a = np.arange(10000000,dtype=np.int8)
sys.getsizeof(a)
```

[13]: 10000104

[ ]:

```
# convenience

a= [i for i in range (10000000)]
b= [i for i in range (10000000, 20000000)]
c=[]

for i in range(len(a)):
    c.append(a[i]+b[i])


import numpy as np
a= np.arange(10000000)
b= np.arange(10000000, 20000000)
c= a+b
```

1.2 Advanced Indexing

[14]:

```
# Normal Indexing and slicing

a = np.arange(24).reshape(6,4)
a
```

[14]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
        [20, 21, 22, 23]])
```

```
[15]: a[1,2]
```

```
[15]: 6
```

```
[16]: a[1:3,1:3]
```

```
[16]: array([[ 5,  6],
             [ 9, 10]])
```

```
[17]: # Fancy Indexing will give the desired row or columns we want which we can't get from normal indexing
      a
```

```
[17]: array([[ 0,  1,  2,  3],
             [ 4,  5,  6,  7],
             [ 8,  9, 10, 11],
             [12, 13, 14, 15],
             [16, 17, 18, 19],
             [20, 21, 22, 23]])
```

```
[19]: a[[0,2,3,5]]
```

```
[19]: array([[ 0,  1,  2,  3],
             [ 8,  9, 10, 11],
             [12, 13, 14, 15],
             [20, 21, 22, 23]])
```

```
[20]: a
```

```
[20]: array([[ 0,  1,  2,  3],
             [ 4,  5,  6,  7],
             [ 8,  9, 10, 11],
             [12, 13, 14, 15],
             [16, 17, 18, 19],
             [20, 21, 22, 23]])
```

```
[21]: a[:,[0,2,3]]
```

```
[21]: array([[ 0,  2,  3],
             [ 4,  6,  7],
             [ 8, 10, 11],
             [12, 14, 15],
             [16, 18, 19],
             [20, 22, 23]])
```

[22]:
```
# Boolean Indexing is used to perform operations on arrays or dataframes
a = np.random.randint(1,100,24).reshape(6,4)
a
```

[22]:
```
array([[36, 47, 66, 55],
       [26, 71, 53, 11],
       [ 4, 48, 77,  3],
       [73, 65, 97, 71],
       [52, 97, 70, 53],
       [82, 19, 59, 14]])
```

[23]:
```
# find all numbers greater than 50
a[a > 50]
```

[23]:
```
array([66, 55, 71, 53, 77, 73, 65, 97, 71, 52, 97, 70, 53, 82, 59])
```

[24]:
```
# find out even numbers
a[a % 2 == 0]
```

[24]:
```
array([36, 66, 26,  4, 48, 52, 70, 82, 14])
```

[25]:
```
# find all numbers greater than 50 and are even

a[(a > 50) & (a % 2 == 0)]
```

[25]:
```
array([66, 52, 70, 82])
```

[26]:
```
# find all numbers not divisible by 7
a[~(a % 7 == 0)]
```

[26]:
```
array([36, 47, 66, 55, 26, 71, 53, 11,  4, 48,  3, 73, 65, 97, 71, 52, 97,
       53, 82, 19, 59])
```

[27]:
```
# find all numbers divisible by 7
a[a%7==0]
```

[27]:
```
array([77, 70, 14])
```

1.3 Broadcasting

The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.

The smaller array is "broadcast" across the larger array so that they have compatible shapes.

[28]:
```
# same shape
a = np.arange(6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)
```

```
print(a)
print(b)

print(a+b)
```

```
[[0 1 2]
 [3 4 5]]
[[ 6  7  8]
 [ 9 10 11]]
[[ 6  8 10]
 [12 14 16]]
```

[29]:

```
# diff shape
a = np.arange(6).reshape(2,3)
b = np.arange(3).reshape(1,3)

print(a)
print(b)

print(a+b)
```

```
[[0 1 2]
 [3 4 5]]
[[0 1 2]]
[[0 2 4]
 [3 5 7]]
```

1.4 Broadcasting Rules

1. Make the two arrays have the same number of dimensions.

- If the numbers of dimensions of the two arrays are different, add new dimensions with size 1 to the head of the array with the smaller dimension.

2. Make each dimension of the two arrays the same size.

- If the sizes of each dimension of the two arrays do not match, dimensions with size 1 are stretched to the size of the other array.
- If there is a dimension whose size is not 1 in either of the two arrays, it cannot be broadcasted, and an error is raised.

[ ]:

```

```

[31]:

```
# More examples

a = np.arange(12).reshape(4,3)
b = np.arange(3)

```

```
print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
[0 1 2]
[[ 0  2  4]
 [ 3  5  7]
 [ 6  8 10]
 [ 9 11 13]]
```

[32]:

```
a = np.arange(12).reshape(3,4)
b = np.arange(3)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[0 1 2]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9756/3056947472.py in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (3,4) (3,)
```

[33]:

```
a = np.arange(3).reshape(1,3)
b = np.arange(3).reshape(3,1)

print(a)
print(b)

print(a+b)
```

```
[[0 1 2]]
[[0]
```

```
 [1]
 [2]]
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```

[34]:

```
a = np.arange(3).reshape(1,3)
b = np.arange(4).reshape(4,1)

print(a)
print(b)

print(a + b)
```

```
[[0 1 2]]
[[0]
 [1]
 [2]
 [3]]
[[0 1 2]
 [1 2 3]
 [2 3 4]
 [3 4 5]]
```

[35]:

```
a = np.array([1])
# shape -> (1,1)
b = np.arange(4).reshape(2,2)
# shape -> (2,2)

print(a)
print(b)

print(a+b)
```

```
[1]
[[0 1]
 [2 3]]
[[1 2]
 [3 4]]
```

[36]:

```
a = np.arange(12).reshape(3,4)
b = np.arange(12).reshape(4,3)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9756/3679270168.py in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (3,4) (4,3)
```

[37]:

```
a = np.arange(16).reshape(4,4)
b = np.arange(4).reshape(2,2)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[0 1]
 [2 3]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9756/1506314148.py in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (4,4) (2,2)
```

1.5 Working with mathematical formulas

```
[38]: a = np.arange(10)
      np.sin(a)
```

```
[38]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
             -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

```
[39]: # sigmoid
      def sigmoid(array):
          return 1/(1 + np.exp(-(array)))


      a = np.arange(100)

      sigmoid(a)
```

```
[39]: array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
             0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661,
             0.9999546 , 0.9999833 , 0.99999386, 0.99999774, 0.99999917,
             0.99999969, 0.99999989, 0.99999996, 0.99999998, 0.99999999,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ,
             1.        , 1.        , 1.        , 1.        , 1.        ])
```

```
[40]: # mean squared error

      actual = np.random.randint(1,50,25)
      predicted = np.random.randint(1,50,25)
```

```
[41]: def mse(actual,predicted):
          return np.mean((actual - predicted)**2)

      mse(actual,predicted)
```

```
[41]: 474.96
```

```
[42]: # binary cross entropy
```

1.6 Working with missing values

```
[43]: # Working with missing values -> np.nan
      a = np.array([1,2,3,4,np.nan,6])
      a
```

```
[43]: array([ 1.,  2.,  3.,  4., nan,  6.])
```

```
[44]: a[~np.isnan(a)]
```

```
[44]: array([1., 2., 3., 4., 6.])
```

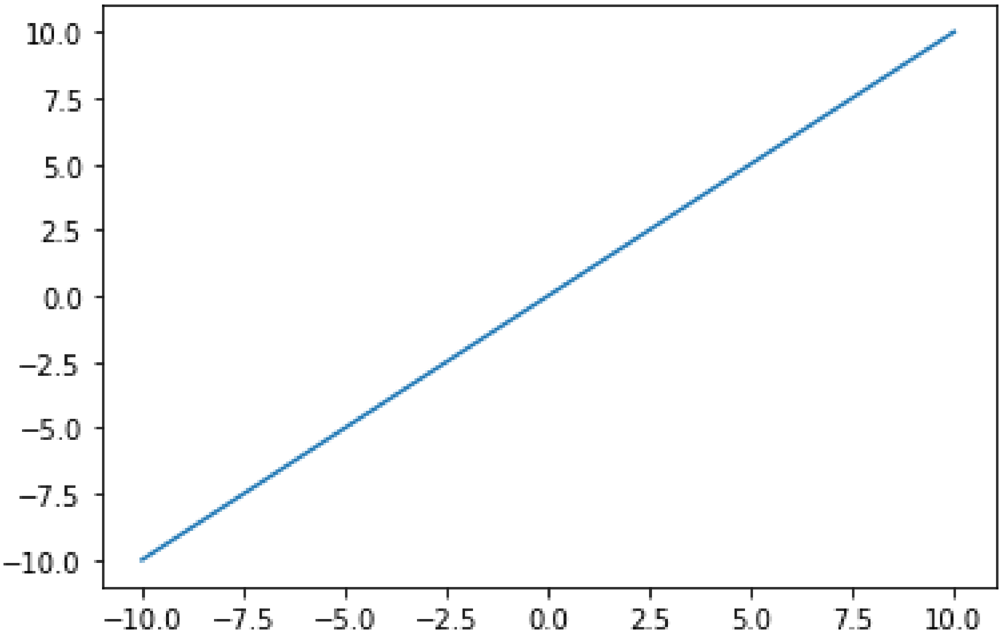
1.7 Plotting Graphs

```
[45]: # plotting a 2D plot
      # x = y
      import matplotlib.pyplot as plt

      x = np.linspace(-10,10,100)
      y = x

      plt.plot(x,y)
```

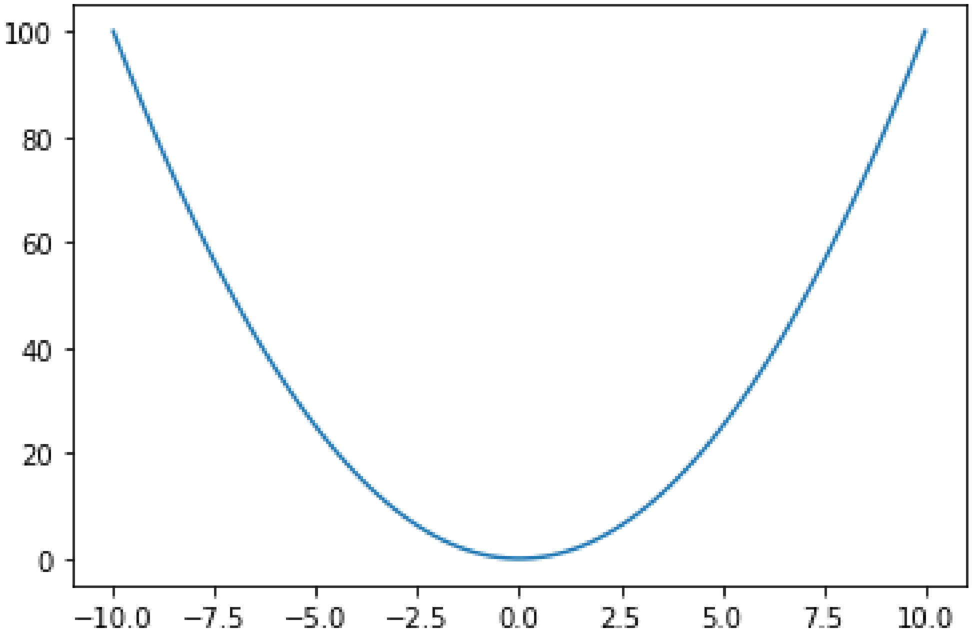
```
[45]: [<matplotlib.lines.Line2D at 0x19b80e99d30>]
```



```
[46]: # y = x^2 Parabola
      x = np.linspace(-10,10,100)
      y = x**2

      plt.plot(x,y)
```

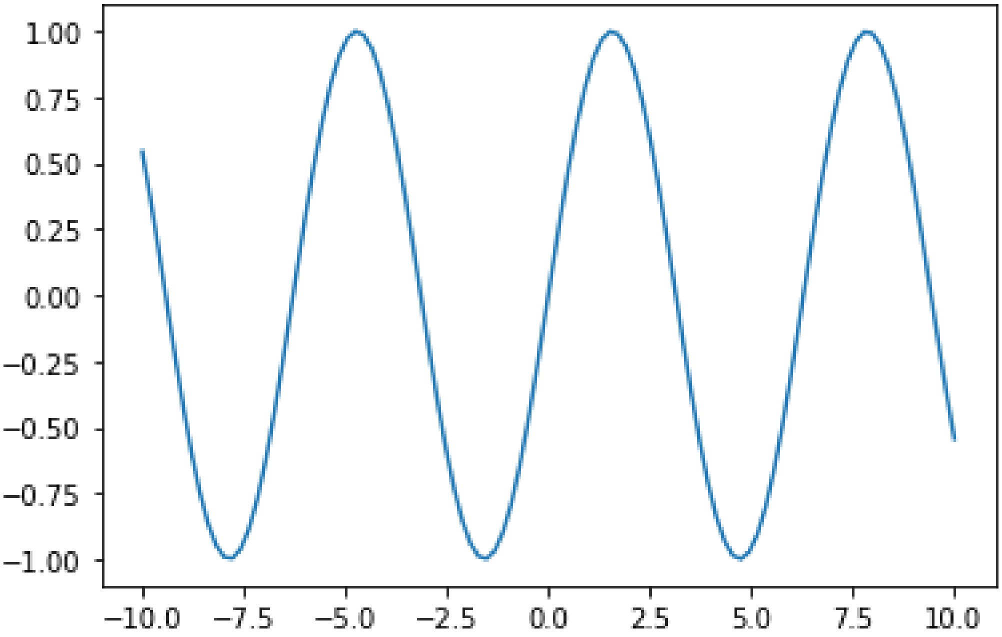
```
[46]: [<matplotlib.lines.Line2D at 0x19b81655a90>]
```



```
[47]: # y = sin(x) Sinusoidal
      x = np.linspace(-10,10,100)
      y = np.sin(x)

      plt.plot(x,y)
```

```
[47]: [<matplotlib.lines.Line2D at 0x19b816d8eb0>]
```

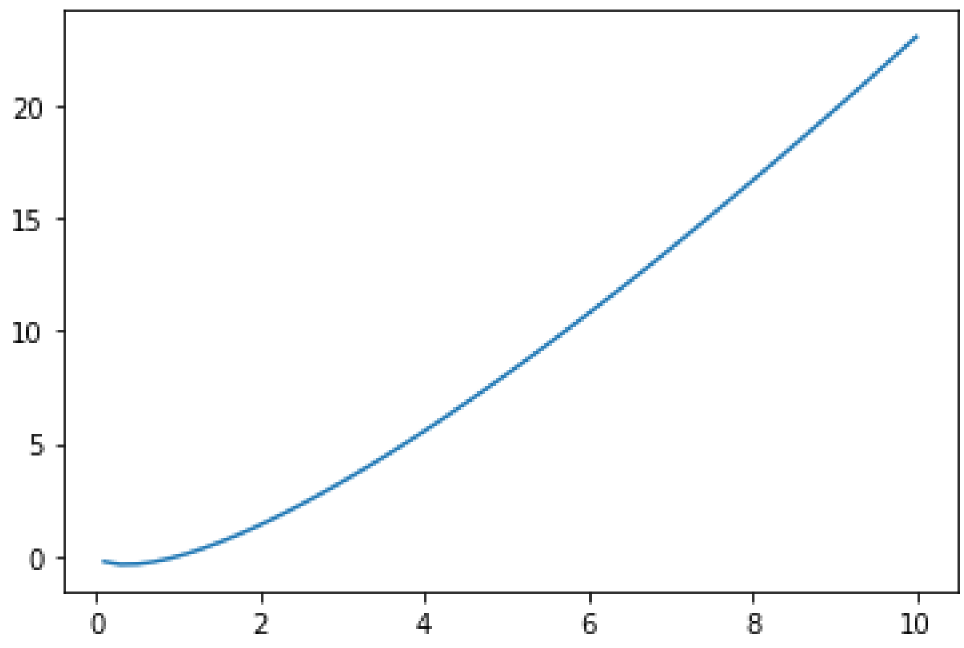


```
[48]: # y = xlog(x) log graph
      x = np.linspace(-10,10,100)
      y = x * np.log(x)

      plt.plot(x,y)
```

```
C:\Users\shiva\AppData\Local\Temp/ipykernel_9756/2564014901.py:3:
RuntimeWarning: invalid value encountered in log
  y = x * np.log(x)
```

```
[48]: [<matplotlib.lines.Line2D at 0x19b8176acd0>]
```



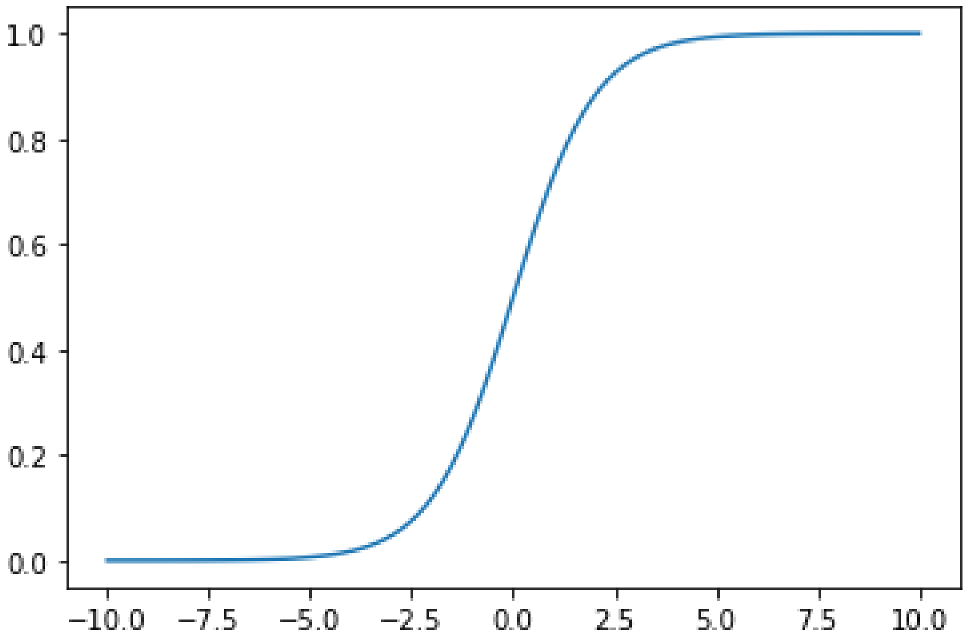
[49]:

```
# sigmoid
x = np.linspace(-10,10,100)
y = 1/(1+np.exp(-x))

plt.plot(x,y)
```

[49]:

```
[<matplotlib.lines.Line2D at 0x19b817e4850>]
```


1.0
0.8
0.6
0.4
0.2
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0