Scalar

In [1]:

```
#Single value
```

In [2]:

```
a =10
print(type(a))
```

```
<class 'int'>
```

In [3]:

```
b=20.8
print(type(b))
```

```
<class 'float'>
```

In [4]:

```
print("Sum")
a + b
```

```
Sum
```

Out[4]:

```
30.8
```

In [5]:

```
print("Sub")
a - b
```

```
Sub
```

Out[5]:

```
-10.8
```

In [6]:

```
print("Div")
a / b
```

```
Div
```

Out[6]:

```
0.4807692307692307
```

In [7]:

```
print("Mul")
a * b
```

```
Mul
```

Out[7]:

```
208.0
```

Vector

In [8]:

```
# Vector is 1-D array, 3-D vector
# Collection of scalar
```

In [9]:

```
#Dimension = Rank,axis
```

In [10]:

```
import numpy as np
np.ndim(a)
```

Out[10]:

```
0
```

In [11]:

```
a = (3,3,5)
b = (6,6,8)
```

In [12]:

```
# a * b
```

In [13]:

```
a + b
```

Out[13]:

```
(3, 3, 5, 6, 6, 8)
```

In [14]:

```
np.add(a,b)
```

Out[14]:

```
array([ 9,  9, 13])
```

In [15]:

```
np.multiply(a,b)
```

Out[15]:

```
array([18, 18, 40])
```

In [16]:

```
np.cross(a,b)
```

Out[16]:

```
array([-6,  6,  0])
```

In [17]:

```
np.dot(a,b)
```

Out[17]:

```
76
```

Matrices

In [18]:

```
#2-D array and 2-D tensor
```

In [19]:

```
x = np.array([[1,4,8],[4,8,1],[5,9,3]])
```

In [20]:

```
x
```

Out[20]:

```
array([[1, 4, 8],
       [4, 8, 1],
       [5, 9, 3]])
```

In [21]:

```
#Matrix and vector multipliction
```

In [22]:

```
s= a * x
```

In [23]:

```
s.shape
```

Out[23]:

```
(3, 3)
```

In [24]:

```
# Matrix and matrix multiplication
```

In [25]:

```
y = np.array([[4,2,1],[5,3,1],[5,1,3]])
```

In [26]:

```
y
```

Out[26]:

```
array([[4, 2, 1],
       [5, 3, 1],
       [5, 1, 3]])
```

In [27]:

```
x * y
```

Out[27]:

```
array([[ 4,  8,  8],
       [20, 24,  1],
       [25,  9,  9]])
```

In [28]:

```
x + y
```

Out[28]:

```
array([[ 5,  6,  9],
       [ 9, 11,  2],
       [10, 10,  6]])
```

In [29]:

```
# Matrix and scalar multiplication
```

In [30]:

```
x * 2
```

Out[30]:

```
array([[ 2,  8, 16],
       [ 8, 16,  2],
       [10, 18,  6]])
```

In [31]:

```
#Transpose of matrix
```

In [32]:

```
np.transpose(y)
```

Out[32]:

```
array([[4, 5, 5],
       [2, 3, 1],
       [1, 1, 3]])
```

Tensor

In [33]:

```
import torch as tc
```

In [ ]:

```
a = tc.Tensor(3,5)
```

In [ ]:

```
b = tc.ones(3,5)
```

In [ ]:

```
tc.mul(a,b)
```

Example

In [43]:

```
# Layer 1
```

In [34]:

```
w = np.array([[0.587,-0.892],[0.588,0.091]])
x = np.array([[0],[1]])
b = np.array([[-2.090],[0.621]])
```

In [35]:

```
w
```

Out[35]:

```
array([[ 0.587, -0.892],
       [ 0.588,  0.091]])
```

In [36]:

```
x
```

Out[36]:

```
array([[0],
       [1]])
```

In [37]:

```
b
```

Out[37]:

```
array([[-2.09 ],
       [ 0.621]])
```

In [38]:

```
y = w*x+b
```

In [39]:

```
y
```

Out[39]:

```
array([[-2.09 , -2.09 ],
       [ 1.209,  0.712]])
```

Binary Classification

In [44]:

```

import numpy as np
from keras.layers import Dense #Node
from keras.models import Sequential # layer
```

In [45]:

```
model = Sequential()
```

In [51]:

```
#units = nodes  #input = input values
model.add(Dense(units=2,activation='relu',input_dim=2) )
model.add(Dense(units=2,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
```

In [53]:

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics='accuracy')
```

In [74]:

```
x = np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
y = np.array([0.,1.,1.,0.])
```

In [75]:

```
x
```

Out[75]:

```
array([[0., 0.],
       [0., 1.],
       [1., 0.],
       [1., 1.]])
```

In [76]:

```
y
```

Out[76]:

```
array([0., 1., 1., 0.])
```

In [77]:

```
'''
batch_size is number of sample processes before the model is updated, suppose that we have
'''
model.fit(x,y,epochs=1000,batch_size=4)
```

```
1/1 [==============================] - 0s 4ms/step - loss: 0.6931 - accura
cy: 0.5000
Epoch 84/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.6931 - accura
cy: 0.5000
Epoch 85/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.6931 - accura
cy: 0.5000
Epoch 86/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.6931
```

In [ ]: