In [39]:
```
import theano
```

In [6]:
```
from theano import tensor as T
```

In [90]:
```
a=T.dscalar()
b=T.dscalar()
res=a-b
func=theano.function([a,b],res)        #converting it to callable object so that it takes matrix as parame
assert 20.0==func(30.5,10.5)
print(func(30.5,10.5))
```

```
20.0
```

In [14]:
```
#addition of two scalars
```

In [92]:
```
x=T.dscalar('x')
y=T.dscalar('y')
z=x+y
f=theano.function([x,y],z)
f(5,7)
```

Out[92]:
```
array(12.)
```

In [91]:
```
#addition of two matrices
```

In [93]:
```
x=T.dmatrix('x')
y=T.dmatrix('y')
z=x+y
f=theano.function([x,y],z)
f([[30,50],[2,3]],[[60,70],[3,4]])
```

Out[93]:
```
array([[ 90., 120.],
       [  5.,   7.]])
```

In [95]:
```
#python program to illustrate logistic
#sigmoid function using theano
a=T.dmatrix('a')
sig=1/(1+T.exp(-a))
log=theano.function([a],sig)
print(log([[0,1],[-1,2]]))
```

```
[[0.5        0.73105858]
 [0.26894142 0.11920292]]
```

In [94]:
```
!pip install torch
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch in c:\users\user\appdata\roaming\python\python39\site-packages (2.0.
1)
Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\lib\site-packages (from torc
h) (4.3.0)
Requirement already satisfied: sympy in c:\programdata\anaconda3\lib\site-packages (from torch) (1.10.1)
Requirement already satisfied: jinja2 in c:\programdata\anaconda3\lib\site-packages (from torch) (2.11.3)
Requirement already satisfied: networkx in c:\programdata\anaconda3\lib\site-packages (from torch) (2.8.
4)
Requirement already satisfied: filelock in c:\programdata\anaconda3\lib\site-packages (from torch) (3.6.
0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\programdata\anaconda3\lib\site-packages (from jinja
2->torch) (2.0.1)
Requirement already satisfied: mpmath>=0.19 in c:\programdata\anaconda3\lib\site-packages (from sympy->to
rch) (1.2.1)
```

In [40]:
```
import torch
```

In [96]:
```
t1=torch.tensor([1,2,3,4])
t2=torch.tensor([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print("Tensor t1 :\n",t1)
print("\n Tensor t2:\n",t2)
#rank of tensors
print("\nRank of t1:",len(t1.shape))
print("\nRank of t2 :",len(t2.shape))
#shape of tensors
print("\n Rank of t1:",t1.shape)
```

```
Tensor t1 :
 tensor([1, 2, 3, 4])

 Tensor t2:
 tensor([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])

Rank of t1: 1

Rank of t2 : 2

 Rank of t1: torch.Size([4])

 Rank of t2: torch.Size([3, 4])
```

In [49]:
```
import numpy as np
```

In [97]:
```
#list of values to be stored as tensor
data1=[1,2,3,4,5,6]
data2=np.array([1.5,3.4,6.8,9.3,7.0,2.8])
#creating tensors and printing
t1=torch.tensor(data1)
t2=torch.tensor(data2)
t3=torch.as_tensor(data2)
t4=torch.from_numpy(data2)
print("Tensor:",t1," Data Type:",t1.dtype,"\n")
print("Tensor:",t2," Data Type:",t2.dtype,"\n")
print("Tensor:",t3," Data Type:",t3.dtype,"\n")
```

```
Tensor: tensor([1, 2, 3, 4, 5, 6])  Data Type: torch.int64

Tensor: tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.8000], dtype=torch.float64)  Data Type: torch.f
loat64

Tensor: tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.8000], dtype=torch.float64)  Data Type: torch.f
loat64

Tensor: tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.8000], dtype=torch.float64)  Data Type: torch.f
loat64
```

In [98]:
```
#defining tensor
t=torch.tensor([[1,2,3,4,],[5,6,7,8],[9,10,11,12]])
#reshaping the tensor
print("Reshaping")
print(t.reshape(6,2))
#Resizing the tensor
print("Resizing")
print(t.resize(2,6))
#transposing the tensor
print("\nTransposing")
print(t.transpose(1,0))
```

```
Reshaping
tensor([[ 1,  2],
        [ 3,  4],
        [ 5,  6],
        [ 7,  8],
        [ 9, 10],
        [11, 12]])
Resizing
tensor([[ 1,  2,  3,  4,  5,  6],
        [ 7,  8,  9, 10, 11, 12]])

Transposing
tensor([[ 1,  5,  9],
        [ 2,  6, 10],
        [ 3,  7, 11],
        [ 4,  8, 12]])
```

In [99]:
```
#defining two tensors
t1=torch.tensor([1,2,3,4])
t2=torch.tensor([5,6,7,8])

#adding two tensors
print("tensor2+tensor1")
print(torch.add(t2,t1))
#substracting two tensors
print("tensor2-tnsor1")
print(torch.sub(t2,t1))
#multipying two tensors
print("tneosr2*tensor1")
print(torch.mul(t2,t1))
#dividing two tensors
print("tensor2/tensor1")
print(torch.div(t2,t1))
```

```
tensor2+tensor1
tensor([ 6,  8, 10, 12])
tensor2-tnsor1
tensor([4, 4, 4, 4])
tneosr2*tensor1
tensor([ 5, 12, 21, 32])
tensor2/tensor1
tensor([5.0000, 3.0000, 2.3333, 2.0000])
```

In [86]:
```
#creating tensors
t1=torch.tensor(1.0,requires_grad=True)
t2=torch.tensor(2.0,requires_grad=True)
#creating a variable and gradient
z=100*t1*t2
z.backward()
#printing gradient
print("dz/dt: ",t1.grad.data)
print("dz/dt: ",t2.grad.data)
```

```
dz/dt:  tensor(200.)
dz/dt:  tensor(100.)
```