201600779 김영민

In [3]:

```
# set some inputs
x =4 ; y = -3;
f = x*y
dfdx = y
dfdy = x
print('dfdx : ',dfdx)
print('dfdy : ',dfdy)
```

```
dfdx :  -3
dfdy :  4
```

In [4]:

```
# set some inputs
x =4 ; y = -3;
f = x+y
dfdx = 1
dfdy = 1
print('dfdx : ',dfdx)
print('dfdy : ',dfdy)
```

```
dfdx :  1
dfdy :  1
```

In [6]:

```
# set some inputs
x =4 ; y = -3;
f = max(x,y)
dfdx = 1
dfdy = 1
print('dfdx : ',dfdx)
print('dfdy : ',dfdy)
```

```
dfdx :  1
dfdy :  1
```

In [9]:

```
# perform the forward pass
x =4 ; y = -3; z= -4;
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
dqdx = 1.0
dqdy = 1.0
# now backprop through q = x + y
dfdx = dfdq * dqdx  # The multiplication here is the chain rule!
dfdy = dfdq * dqdy
print('dfdx : ',dfdx)
print('dfdy : ',dfdy)
```

```
dfdx :  -4.0
dfdy :  -4.0
```

In [14]:

```
import math
w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the sigmoid gradient derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the circuit
print('dx : ',dx)
print('dw : ',dw)
```

```
dx :  [0.3932238664829637, -0.5898357997244456]
dw :  [-0.19661193324148185, -0.3932238664829637, 0.19661193324148185]
```

201600779 김영민