201600282 엄기산

In [22]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import math
import numpy as np
```

In [23]:
```
# 1. f1 = xy
# df1/dx = y, df1/dy = x
x = 4;    y = - 3
f1 = x * y              #then f become -12

df1dx = y               #then dfdx become -3
df1dy = x               #then dfdy become 4
x
y
f1
df1dx
df1dy
```

Out[23]: 4

Out[23]: -3

Out[23]: -12

Out[23]: -3

Out[23]: 4

In [24]:
```
# 2. f2 = x + y
# df2/dx = 1, df2/dy = 1
x = 4;    y = - 3
f2 = x + y              #then f2 become 1

df2dx = 1               #then df2dx become 1
df2dy = 1               #then df2dy become 1
x
y
f2
df2dx
df2dy
```

Out[24]: 4

Out[24]: -3

Out[24]: 1

Out[24]: 1

Out[24]: 1

In [25]:
```
# 3. f3 = max(x, y)
x = 4;    y = - 3
if(x>=y) :
    f3 = x          #since f3 = max(x,y) f3 become 4
    df3dx = 1
    df3dy = 0

else :
```

```
    f3 = y
    df3dx = 0
    df3dy = 1

x
y
f3
df3dx
df3dy
```

Out[25]: 4

Out[25]: -3

Out[25]: 4

Out[25]: 1

Out[25]: 0

In [26]:

```
#4. f4 = (x+y)z
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f4 = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
df4dz = q # df/dz = q, so gradient on z becomes 3
df4dq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
df4dx = 1.0 * df4dq # dq/dx = 1. And the multiplication here is the chain rule!
df4dy = 1.0 * df4dq # dq/dy = 1
x
y
z
f4
df4dx
df4dy
df4dz
df4dq
```

Out[26]: -2

Out[26]: 5

Out[26]: -4

Out[26]: -12

Out[26]: -4.0

Out[26]: -4.0

Out[26]: 3

Out[26]: -4

In [27]:

```
#5. 시그모이드 예제
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
```

```python
# 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

w
x
dot
f
ddot
dx
dw
```

Out[27]: [2, -3, -3]

Out[27]: [-1, -2]

Out[27]: 1

Out[27]: 0.7310585786300049

Out[27]: 0.19661193324148185

Out[27]: [0.3932238664829637, -0.5898357997244456]

Out[27]: [-0.19661193324148185, -0.3932238664829637, 0.19661193324148185]

201600282 엄기산