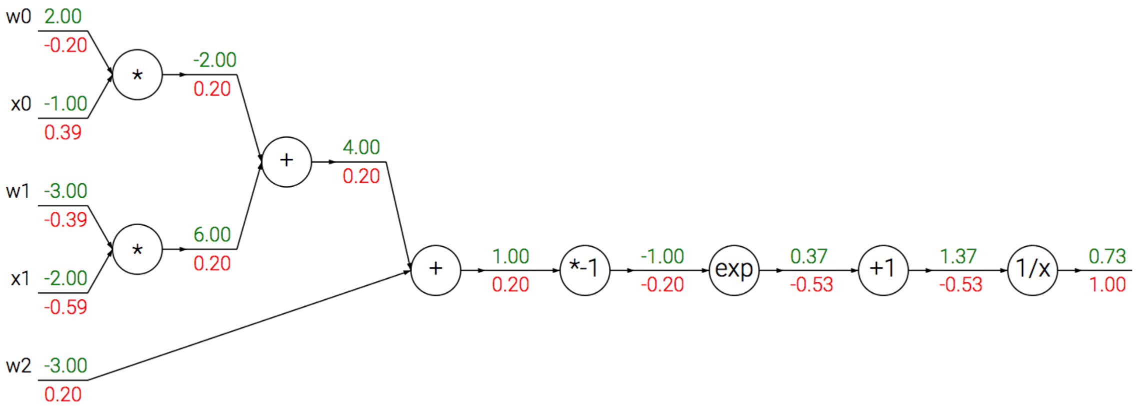
### Backpropagation

# Sigmoid example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\rightarrow \qquad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$




w0=2　w1=-3　w2=-3　x0=-1　x1=-2

绿色的是数值

红色的是导数，如 df/dσ(w0) = -0.20

Lets see the backprop for this neuron in code:

```
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
```

---

## Another complex example

$$f(x, y) = \frac{x + \sigma(y)}{\sigma(x) + (x + y)^2}$$

```
x = 3 # example values
y = -4

# forward pass
sigy = 1.0 / (1 + math.exp(-y)) # sigmoid in numerator   #(1)
num = x + sigy # numerator                               #(2)
sigx = 1.0 / (1 + math.exp(-x)) # sigmoid in denominator #(3)
xpy = x + y                                              #(4)
xpysqr = xpy**2                                          #(5)
den = sigx + xpysqr # denominator                        #(6)
invden = 1.0 / den                                       #(7)
f = num * invden # done!                                 #(8)
```

```
# backprop f = num * invden
dnum = invden # gradient on numerator                             #(8)
dinvden = num                                                     #(8)
# backprop invden = 1.0 / den
ddenn = (-1.0 / (den**2)) * dinvden                               #(7)
# backprop den = sigx + xpysqr
dsigx = (1) * dden                                                #(6)
dxpysqr = (1) * dden                                              #(6)
# backprop xpysqr = xpy**2
```

```
dxpy = (2 * xpy) * dxpysqr                              #(5)
# backprop xpy = x + y
dx = (1) * dxpy                                         #(4)
dy = (1) * dxpy                                         #(4)
# backprop sigx = 1.0 / (1 + math.exp(-x))
dx += ((1 - sigx) * sigx) * dsigx # Notice += !! See notes below  #(3)
# backprop num = x + sigy
dx += (1) * dnum                                        #(2)
dsigy = (1) * dnum                                      #(2)
# backprop sigy = 1.0 / (1 + math.exp(-y))
dy += ((1 - sigy) * sigy) * dsigy                       #(1)
# done! phew
```

- **Cache forward pass variables** 记录已经算过的东西

- **Gradients add up at forks**. The forward expression involves the variables **x,y** multiple times, so when we perform backpropagation we must be careful to use `+=` instead of `=` to accumulate the gradient on these variables (otherwise we would overwrite it). This follows the *multivariable chain rule* in Calculus, which states that if a variable branches out to different parts of the circuit, then the gradients that flow back to it will add.

---

# Matrix-Matrix multiply gradient

```
# forward pass
W = np.random.randn(5, 10)
X = np.random.randn(10, 3)
D = W.dot(X)


# now suppose we had the gradient on D from above in the circuit
dD = np.random.randn(*D.shape) # same shape as D
dW = dD.dot(X.T) #.T gives the transpose of the matrix
dX = W.T.dot(dD)
```