This expression describes a 2-dimensional neuron (with inputs **x** and weights **w**) that uses the sigmoid activation function.

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

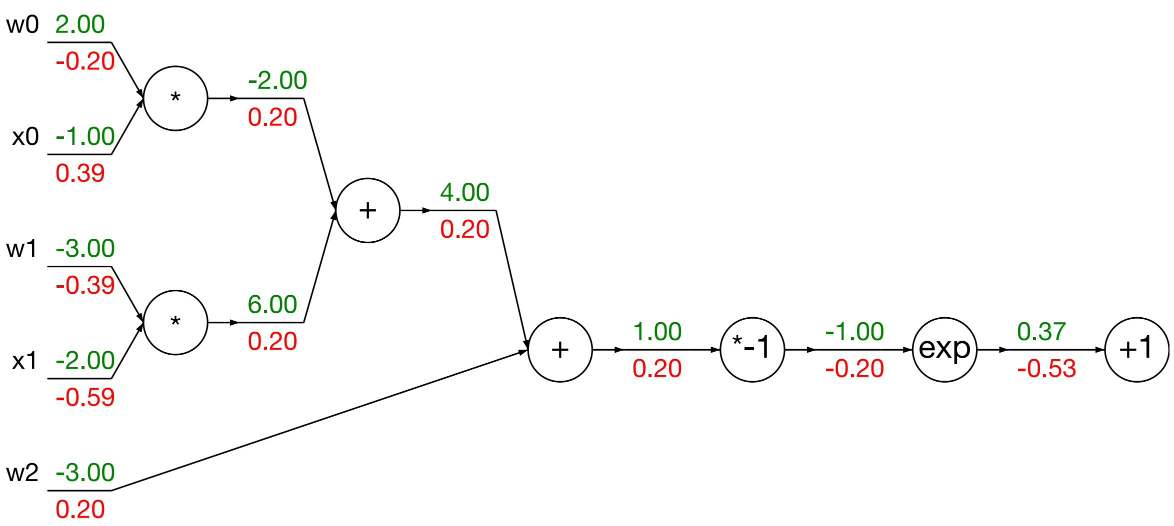
The inputs are [x0,x1] and the (learnable) weights of the neuron are [w0,w1,w2]. The neuron computes a dot product with the input and then its activation is softly squashed by the sigmoid function to be in range from 0 to 1.

$$dot = w[0]*x[0]+w[1]*x[1]+w[2]$$

In [4]:

```
from IPython.display import SVG
def show_svg():
    return SVG('<svg width="799" height="306"><g transform="scale(0.8)"><defs><marke
show_svg()
```

Out[4]:



The sigmoid expression receives the input 1.0 and computes the output 0.73 during the forward pass.

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

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

The local gradient would simply be (1 - 0.73) * 0.73 ~= 0.2

In [10]:

```
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 derivatio
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 'ddot = ', ddot
print
print 'backprop into x'
print 'dx = ', dx
print
print 'backprop into w'
print 'dw = ', dw
```

```
ddot =  0.196611933241

backprop into x
dx =  [0.3932238664829637, -0.5898357997244456]

backprop into w
dw =  [-0.19661193324148185, -0.3932238664829637, 0.19661193324148185]
```

#This page is based on http://cs231n.github.io/optimization-2/ (http://cs231n.github.io/optimization-2/)

#Edited by Seung-Chan Kim