Learning for vision II
Neural networks

Karel Zimmermann
http://cmp.felk.cvut.cz/~zimmerk/

Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

Center for Machine Perception
https://cmp.felk.cvut.cz

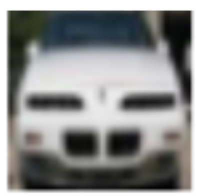
Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Linear classifier and neuron

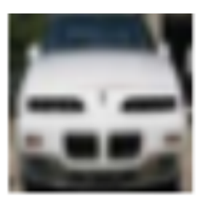
Labels | RGB images

$+1$

-1

def classify():

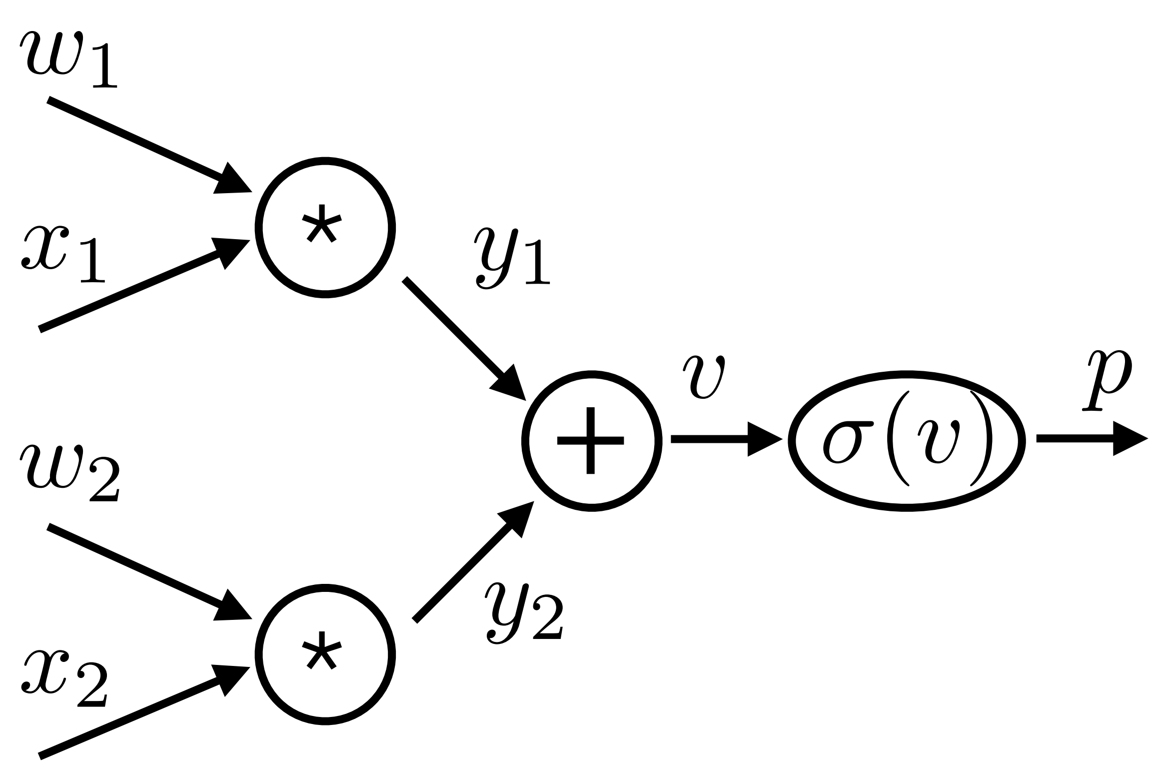
 # Linear classifier

$\mathbf{x} = \text{vec}($  $)$

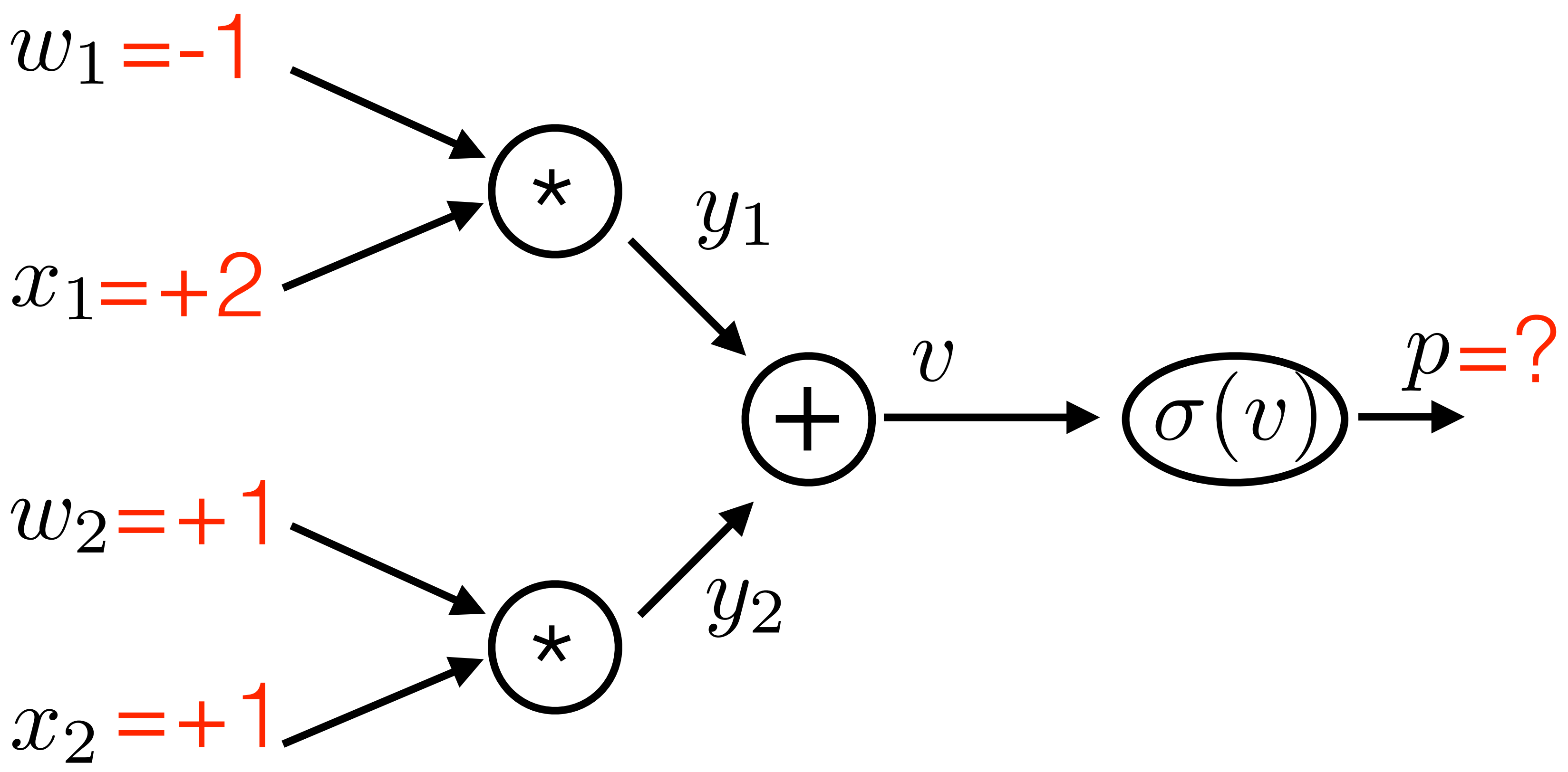
$p = \sigma\left(\mathbf{w}^\top \mathbf{x}\right)$

return p

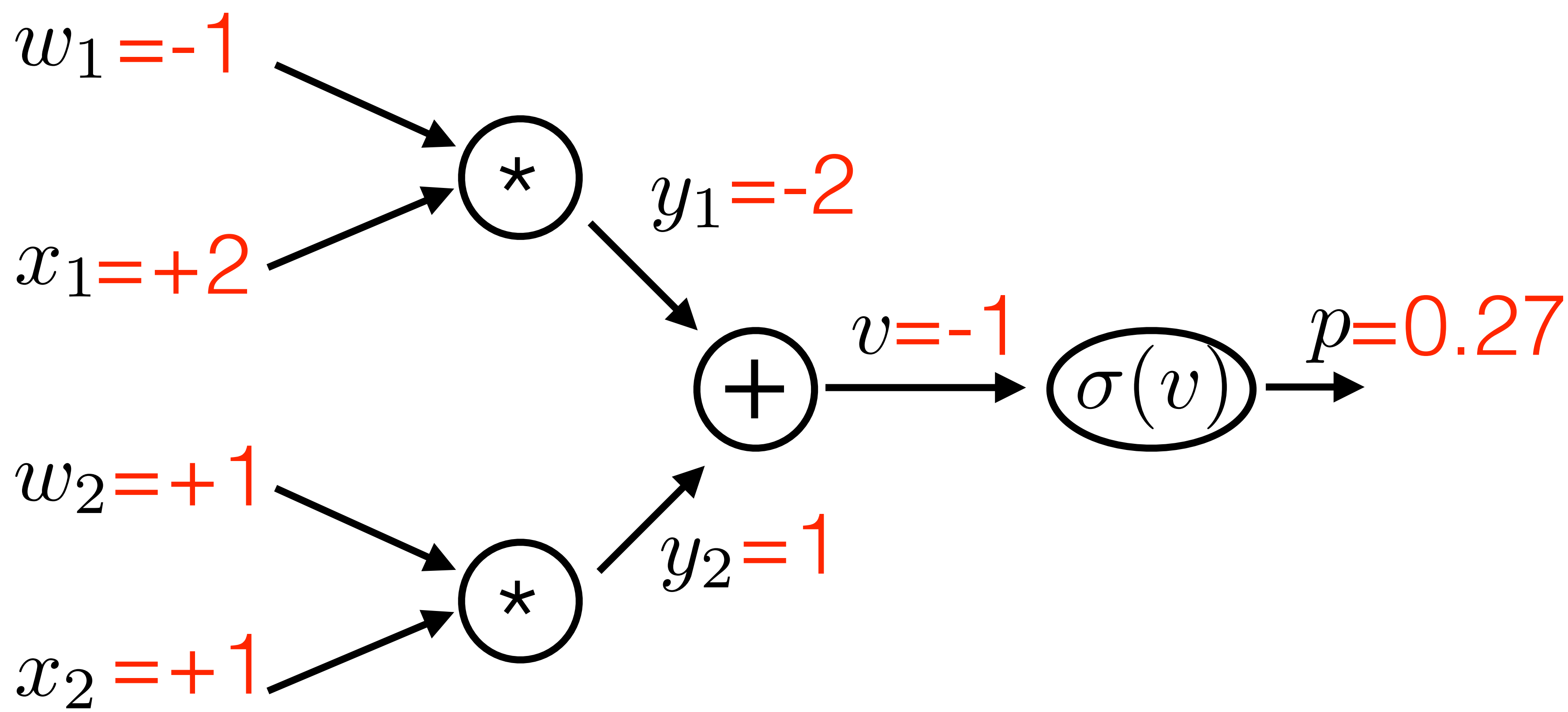
Computational graph of linear classifier



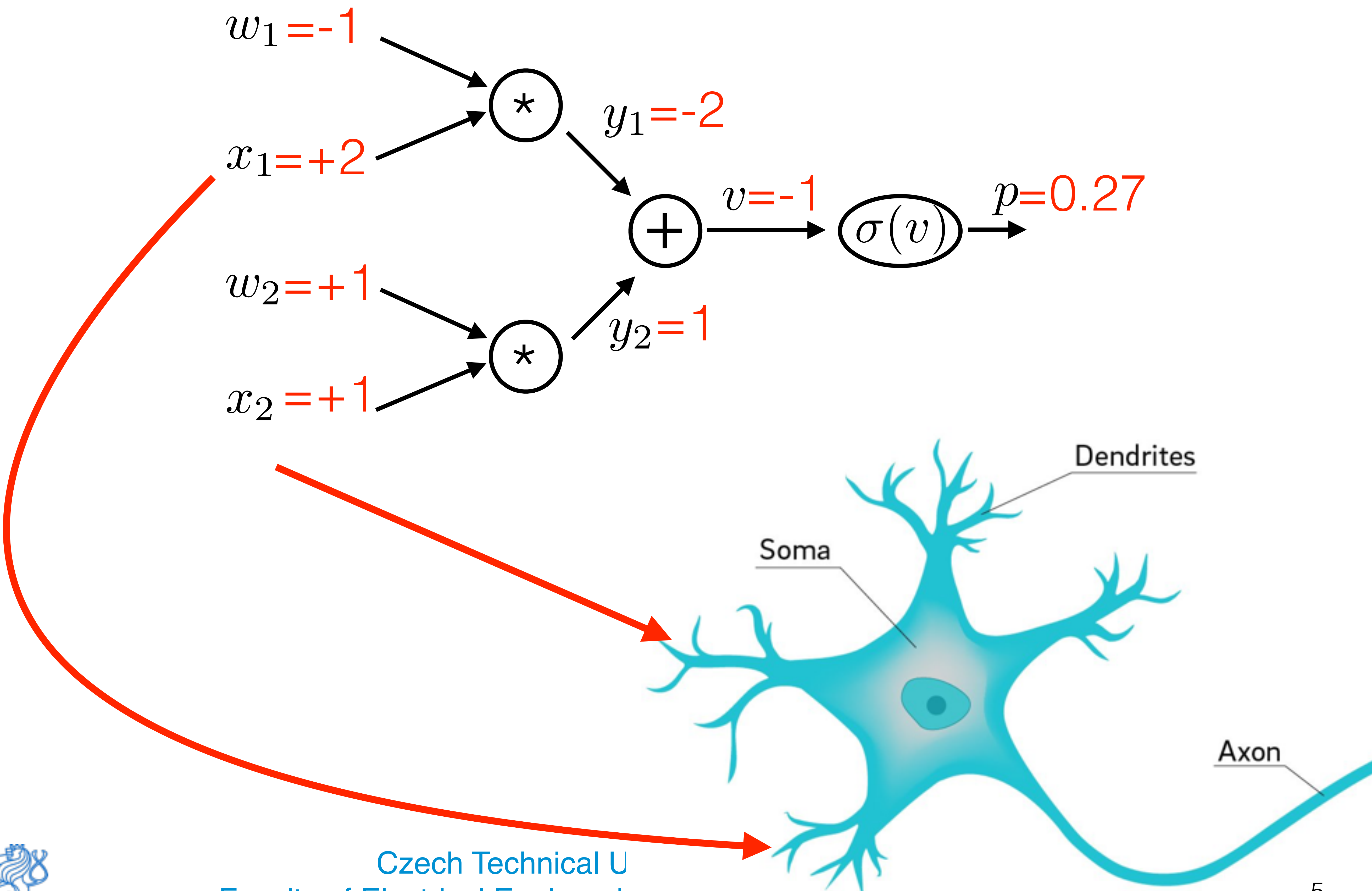
Example I: given trained neuron, and input, what is output?



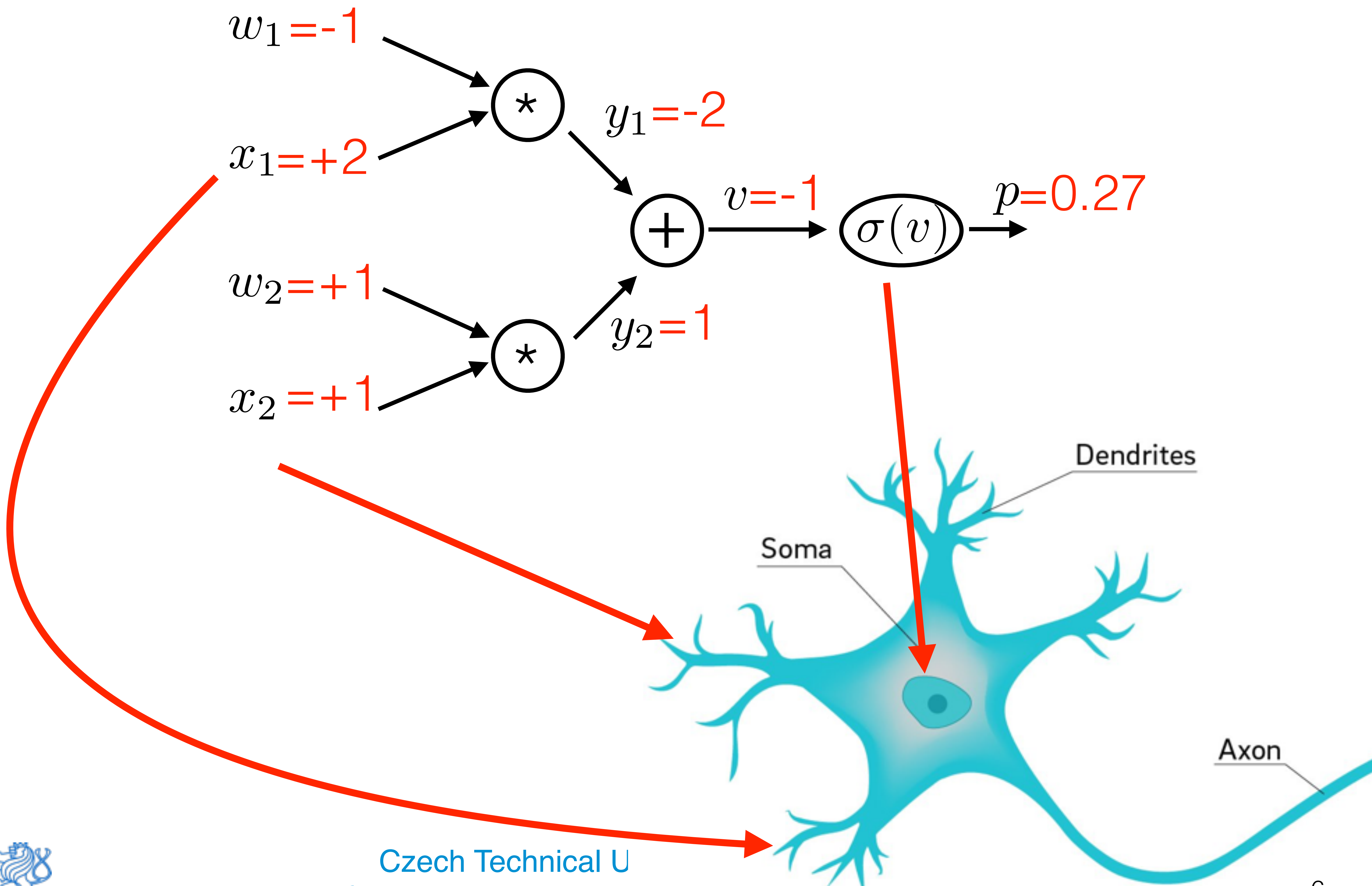
Example I: given trained classifier, and input, what is output?



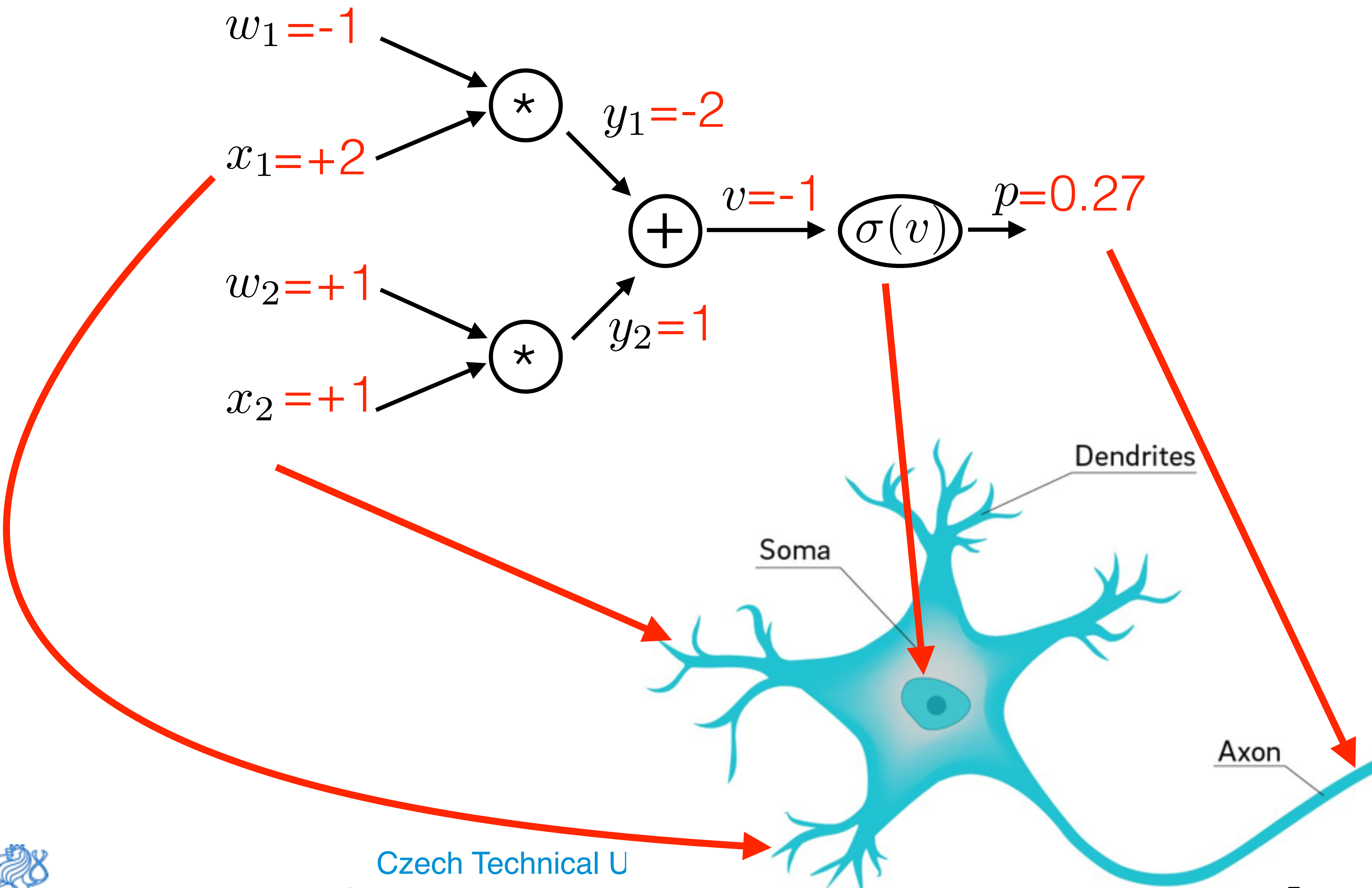
Relation to biological neuron



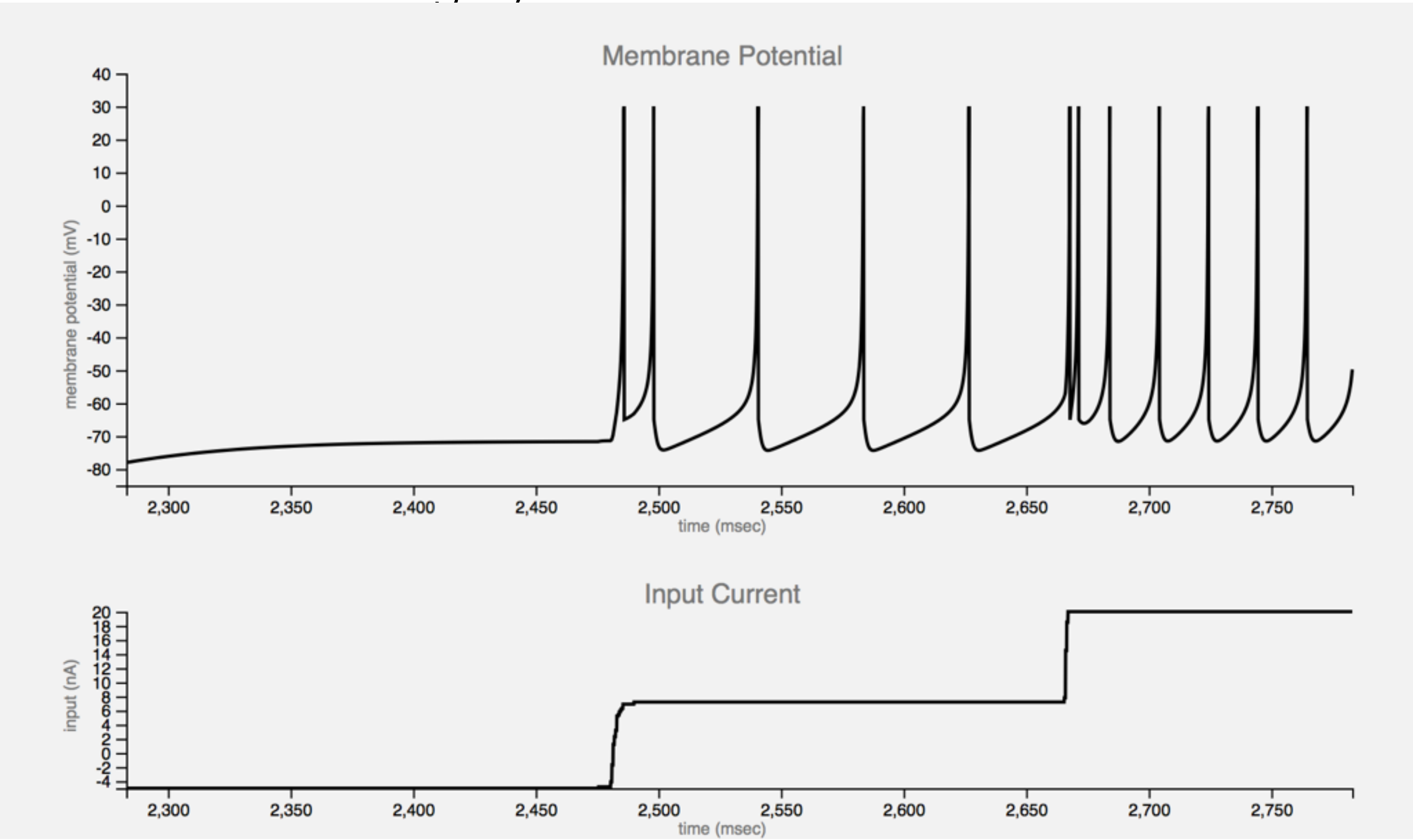
Relation to biological neuron



Relation to biological neuron



Modeling dynamic neuron behaviour

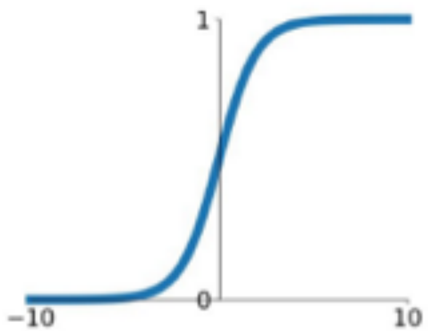


http://jackterwilliger.com/biological-neural-networks-part-i-spiking-neurons/

Activation functions $\sigma(v)$

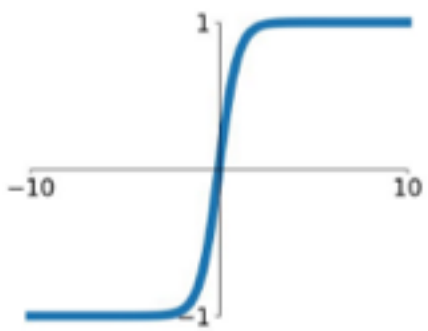
Sigmoid

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



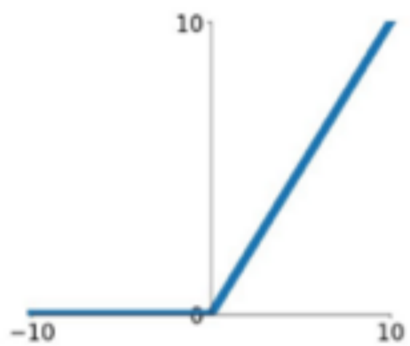
tanh

$\tanh(x)$



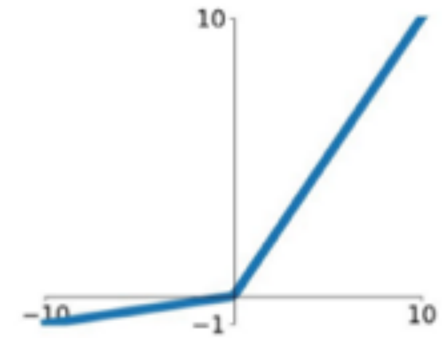
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

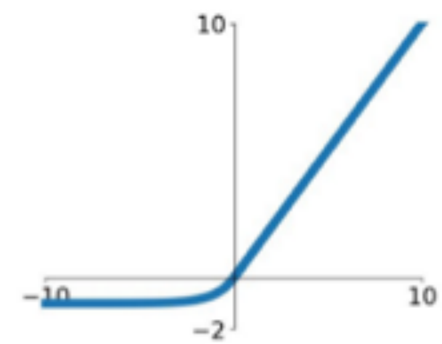


Maxout

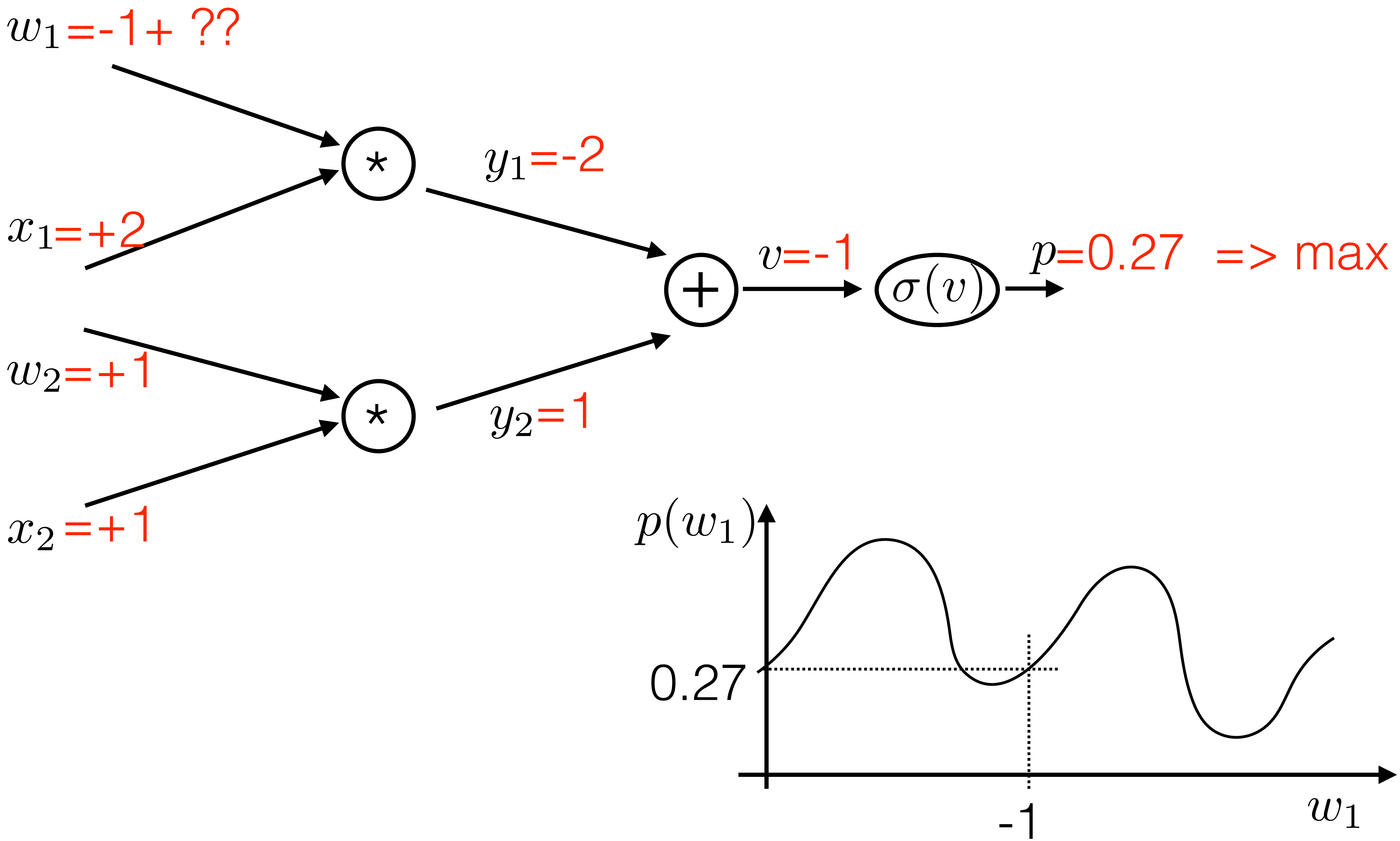
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

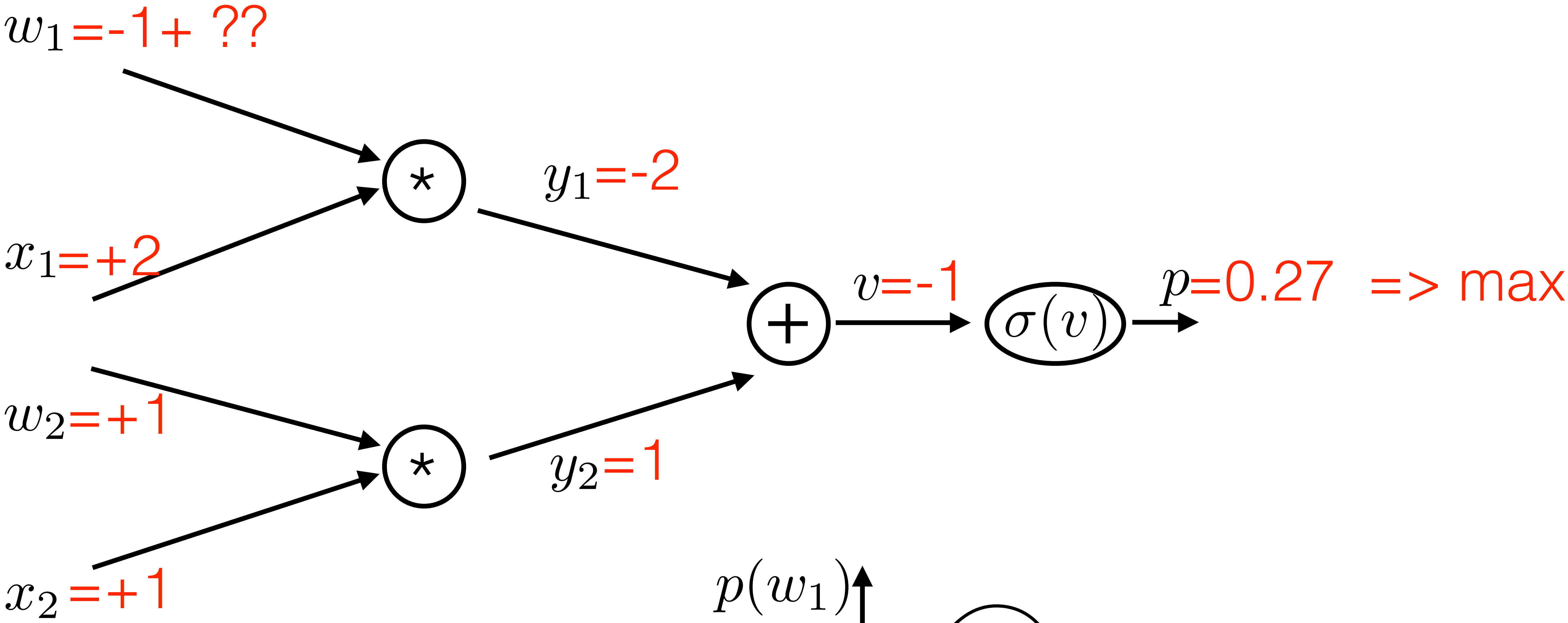
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



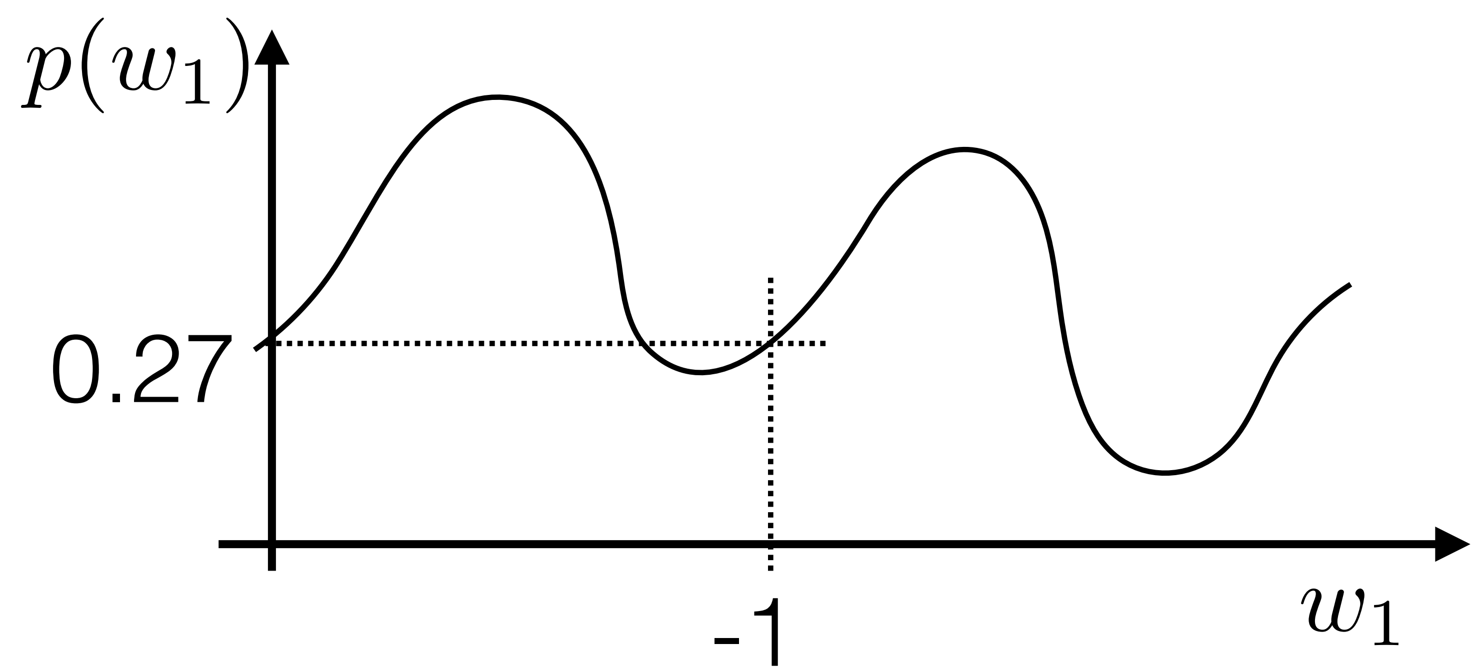
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



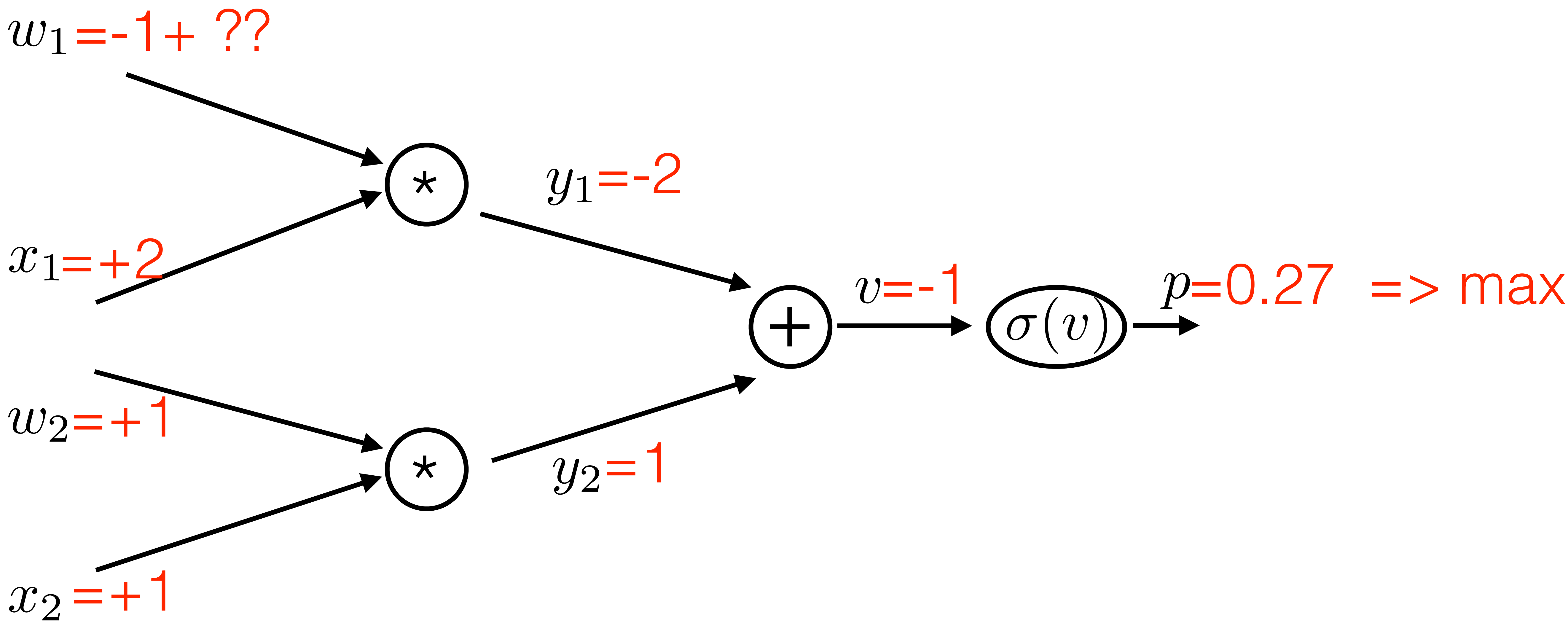
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$



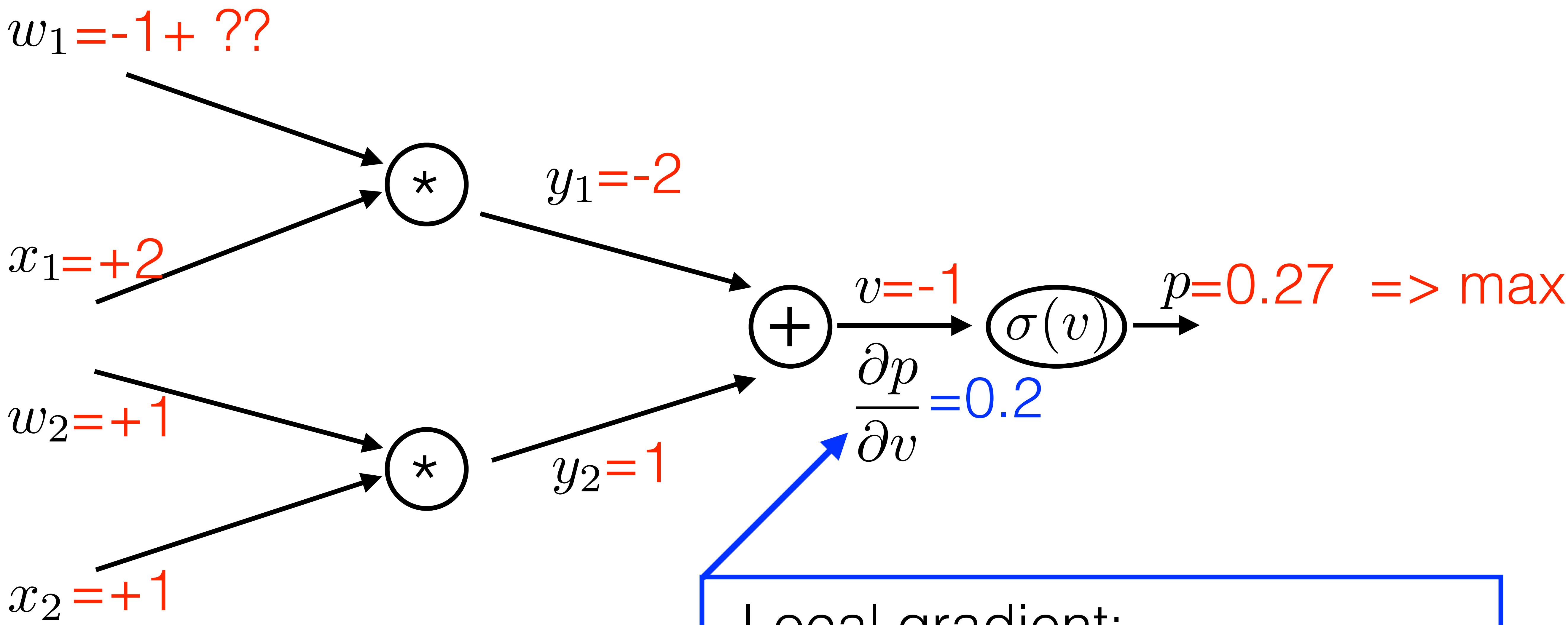
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\dfrac{\partial p}{\partial w_1} = \dfrac{\partial p}{\partial v}\dfrac{\partial v}{\partial y_1}\dfrac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



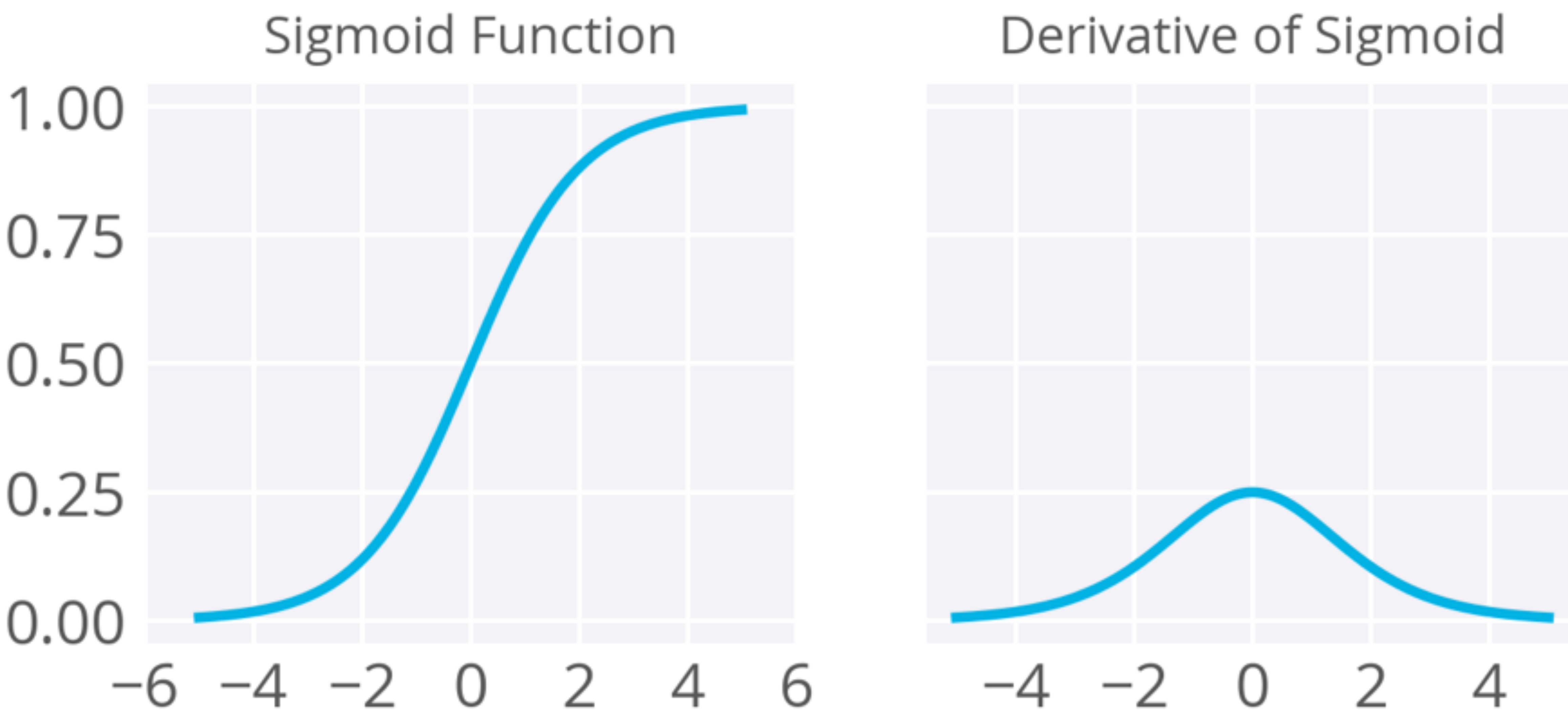
Local gradient:

$$\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1 - \sigma(v))$$

$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$

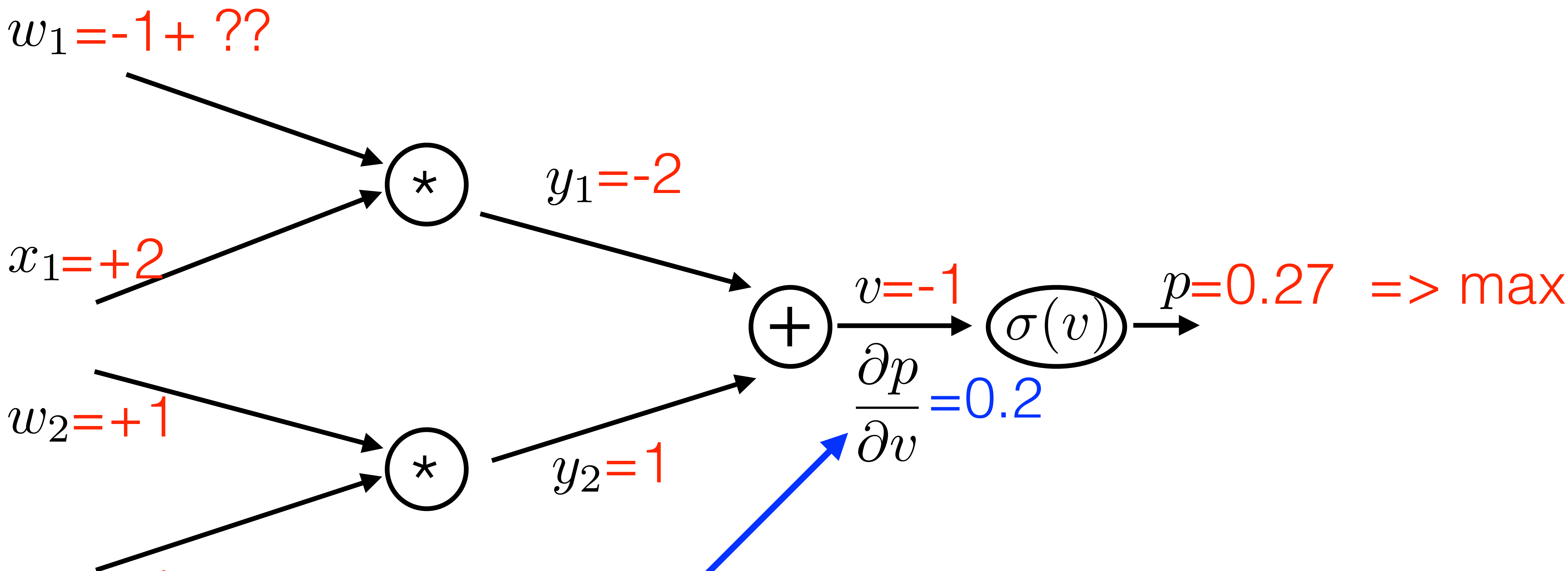
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



Local gradient:

$$\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1 - \sigma(v))$$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



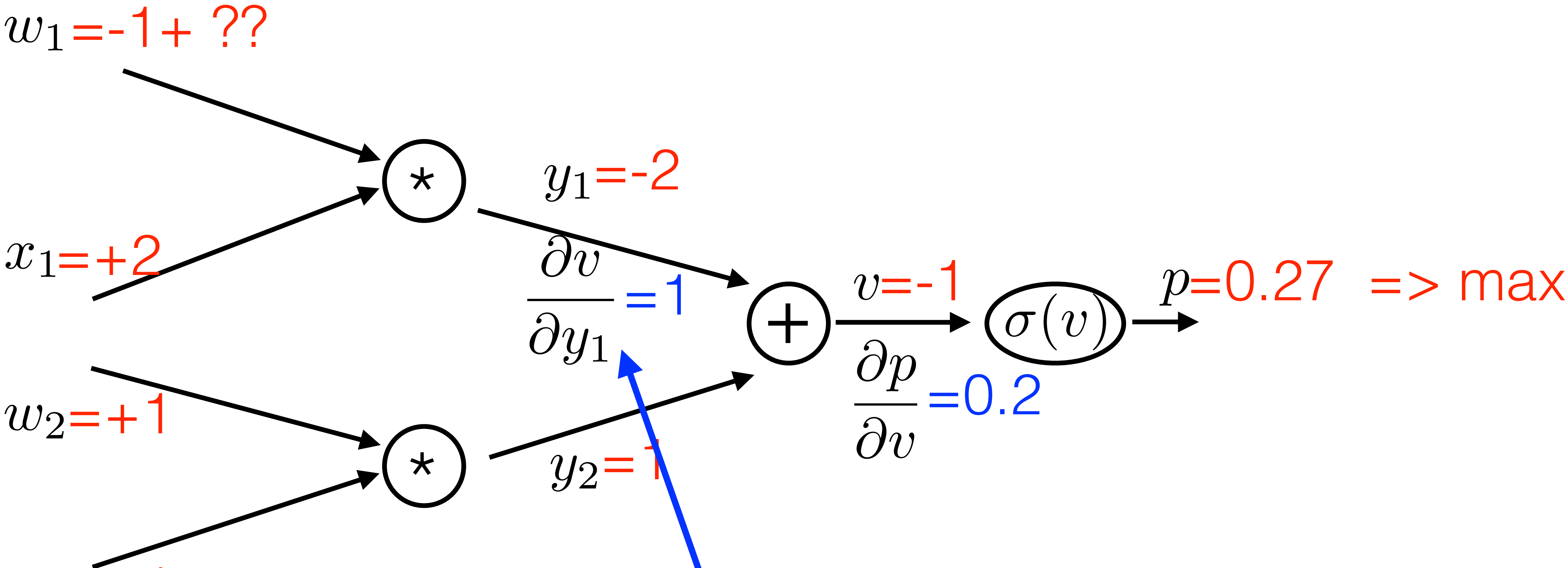
Local gradient:

$$\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1 - \sigma(v))$$

$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v} \frac{\partial v}{\partial y_1} \frac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



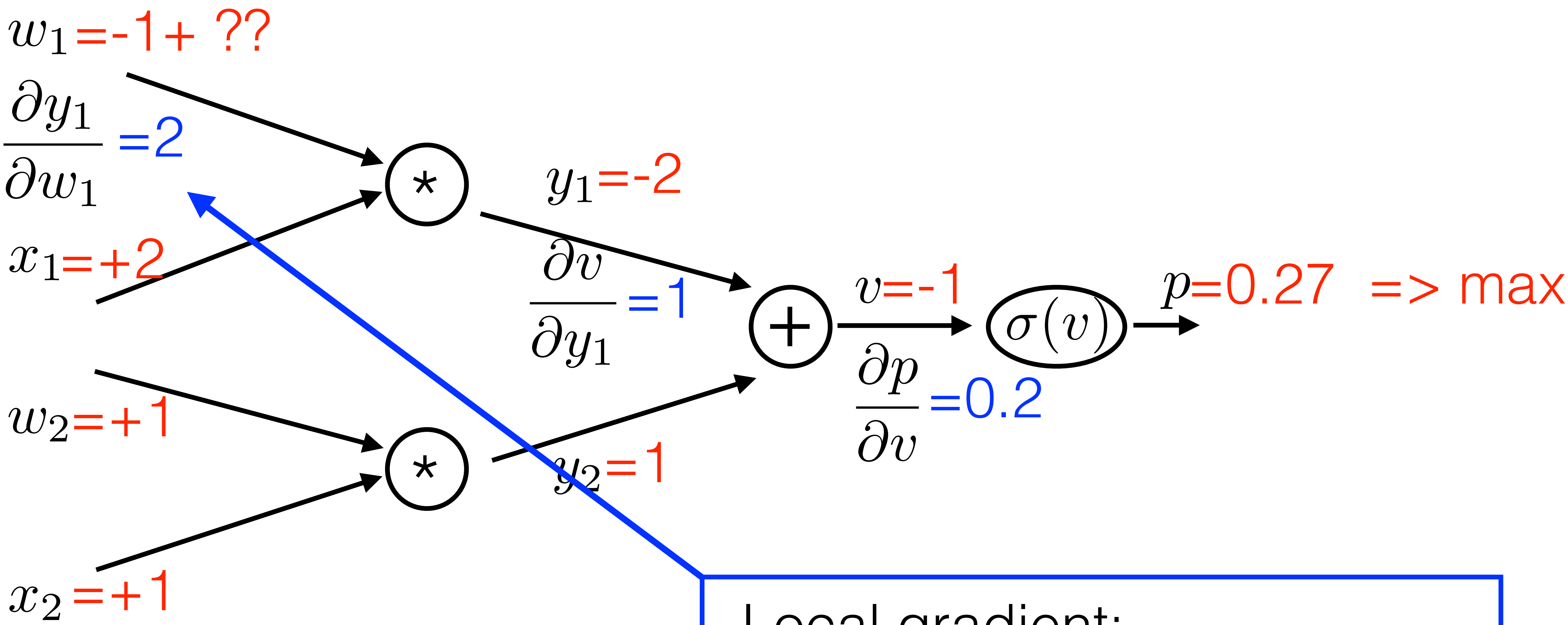
Local gradient:

$$\frac{\partial v}{\partial y_1} = \frac{\partial (y_1 + y_2)}{\partial y_1} = 1$$

$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v} \frac{\partial v}{\partial y_1} \frac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



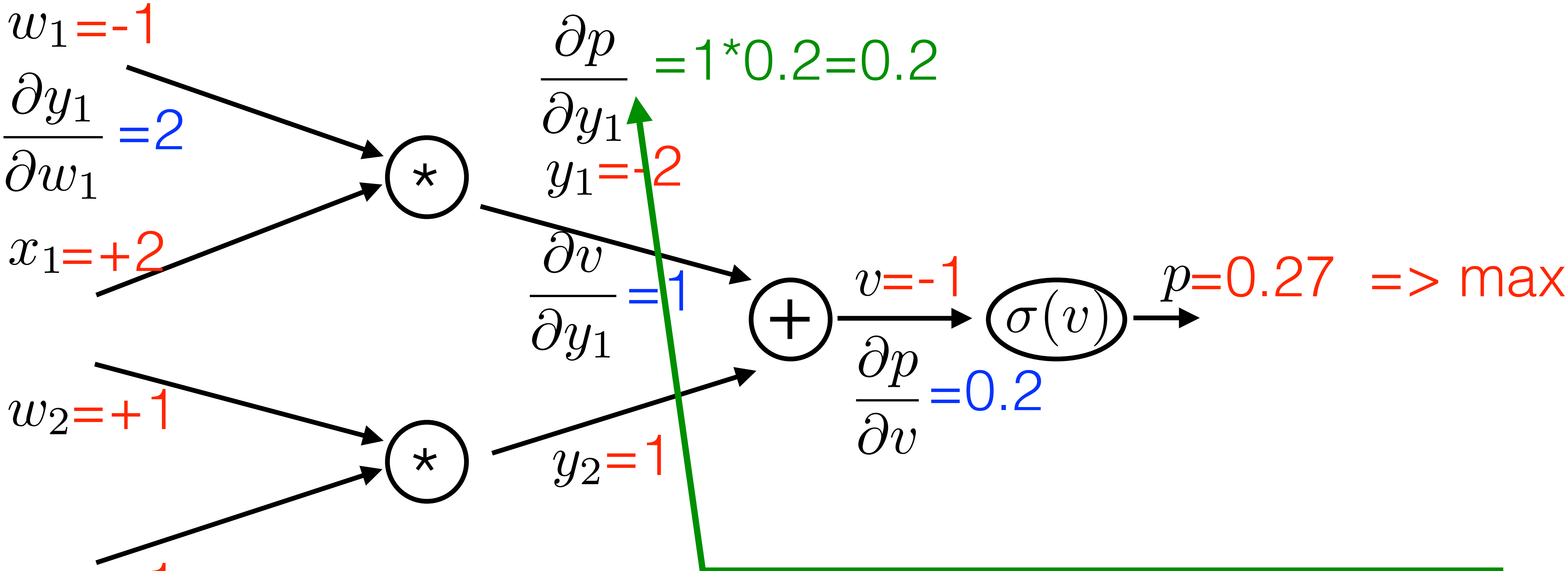
Local gradient:

$$\frac{\partial y_1}{\partial w_1} = \frac{\partial (w_1 x_1)}{\partial w_1} = x_1$$

$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



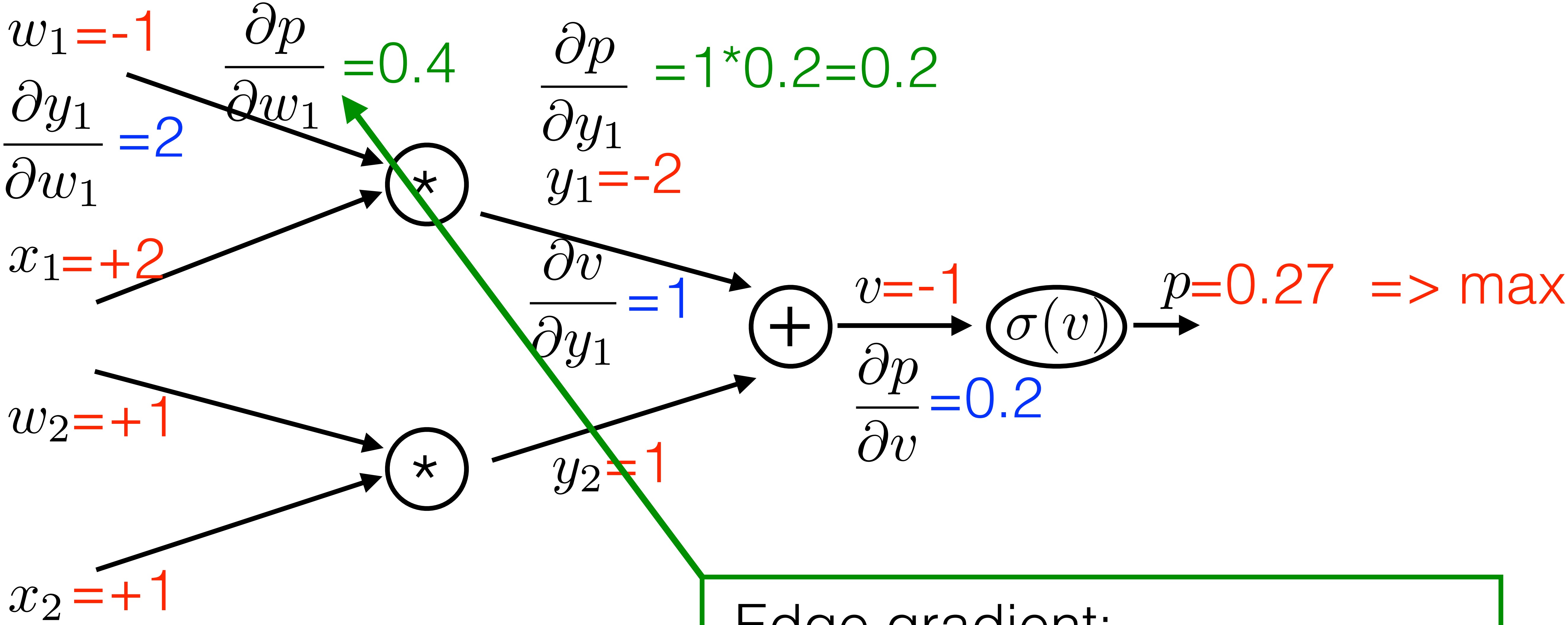
$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Edge gradient:

$$\frac{\partial p}{\partial y_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



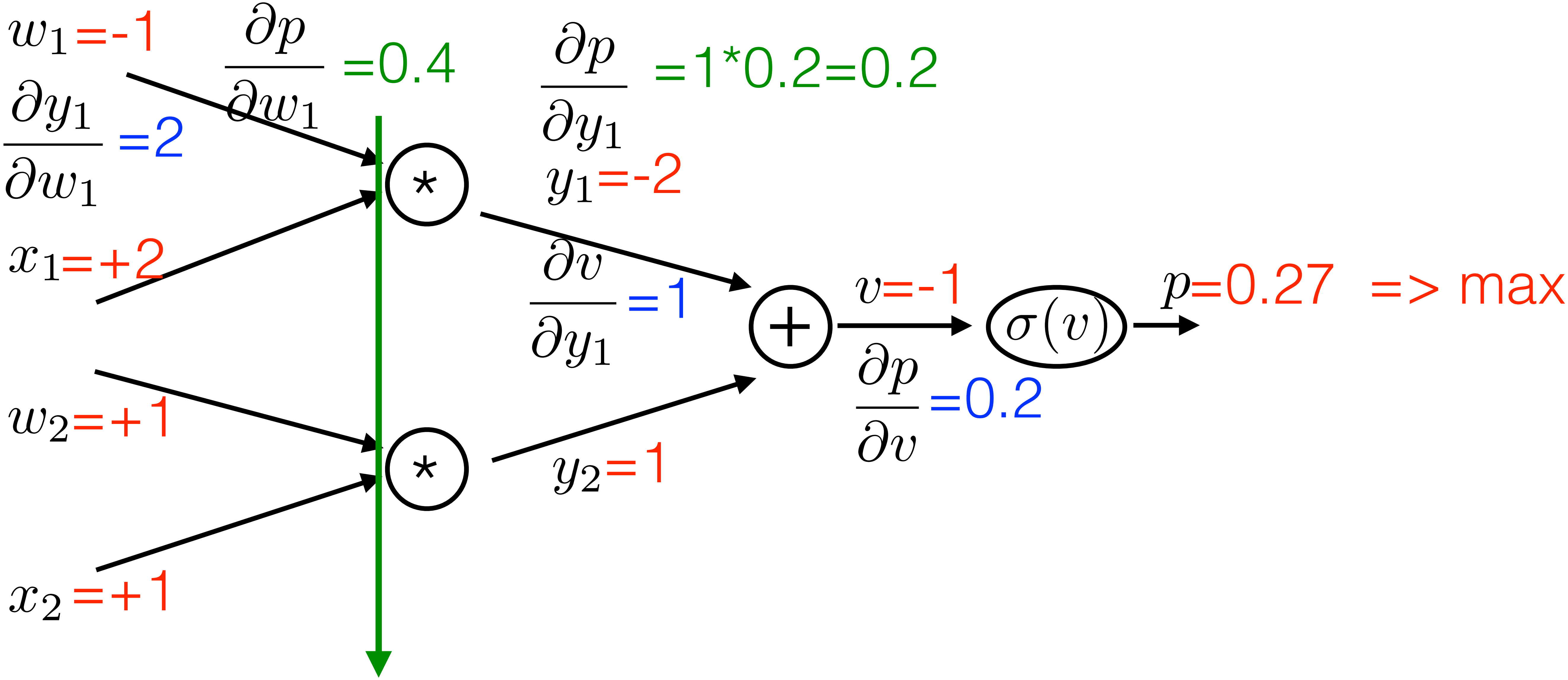
$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Edge gradient:

$$\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial y_1} \frac{\partial y_1}{\partial w_1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v} \frac{\partial v}{\partial y_1} \frac{\partial y_1}{\partial w_1}$

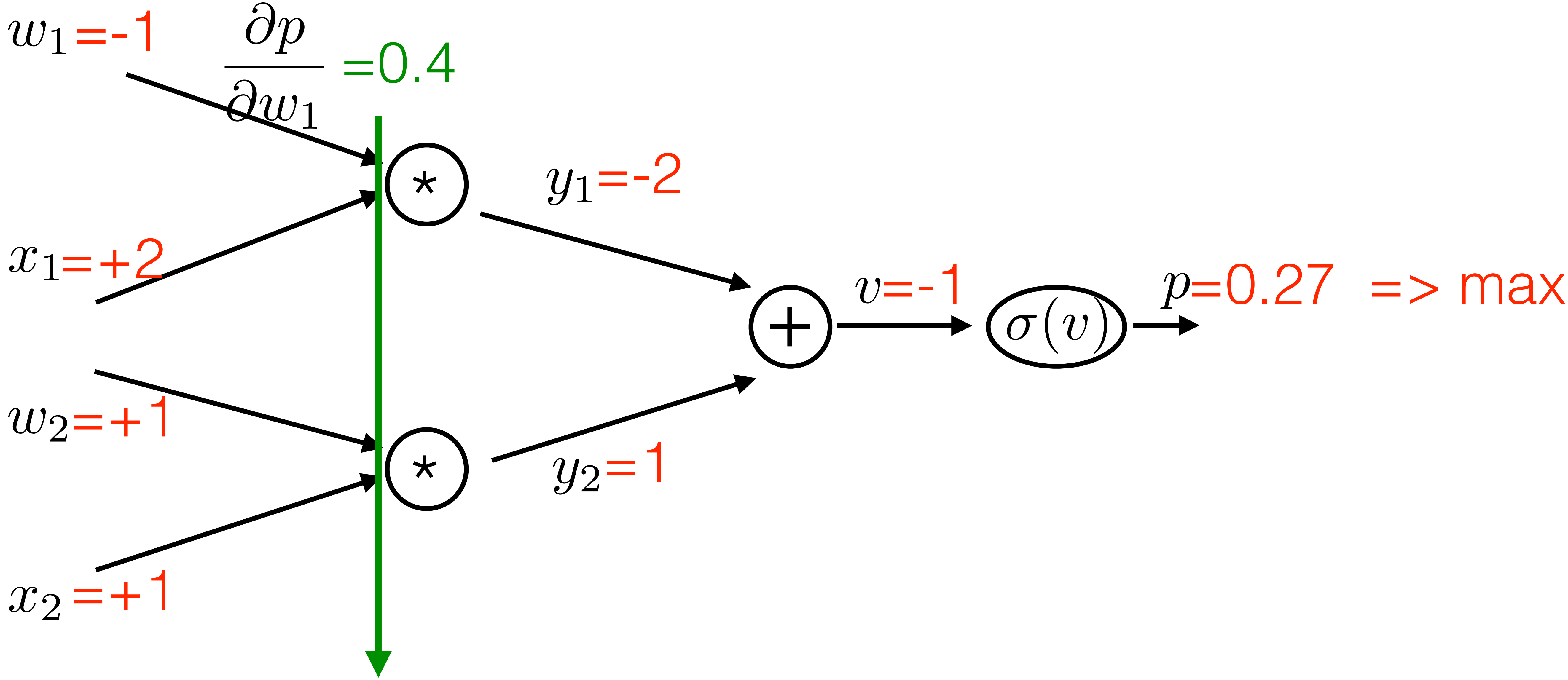
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v} \frac{\partial v}{\partial y_1} \frac{\partial y_1}{\partial w_1}$

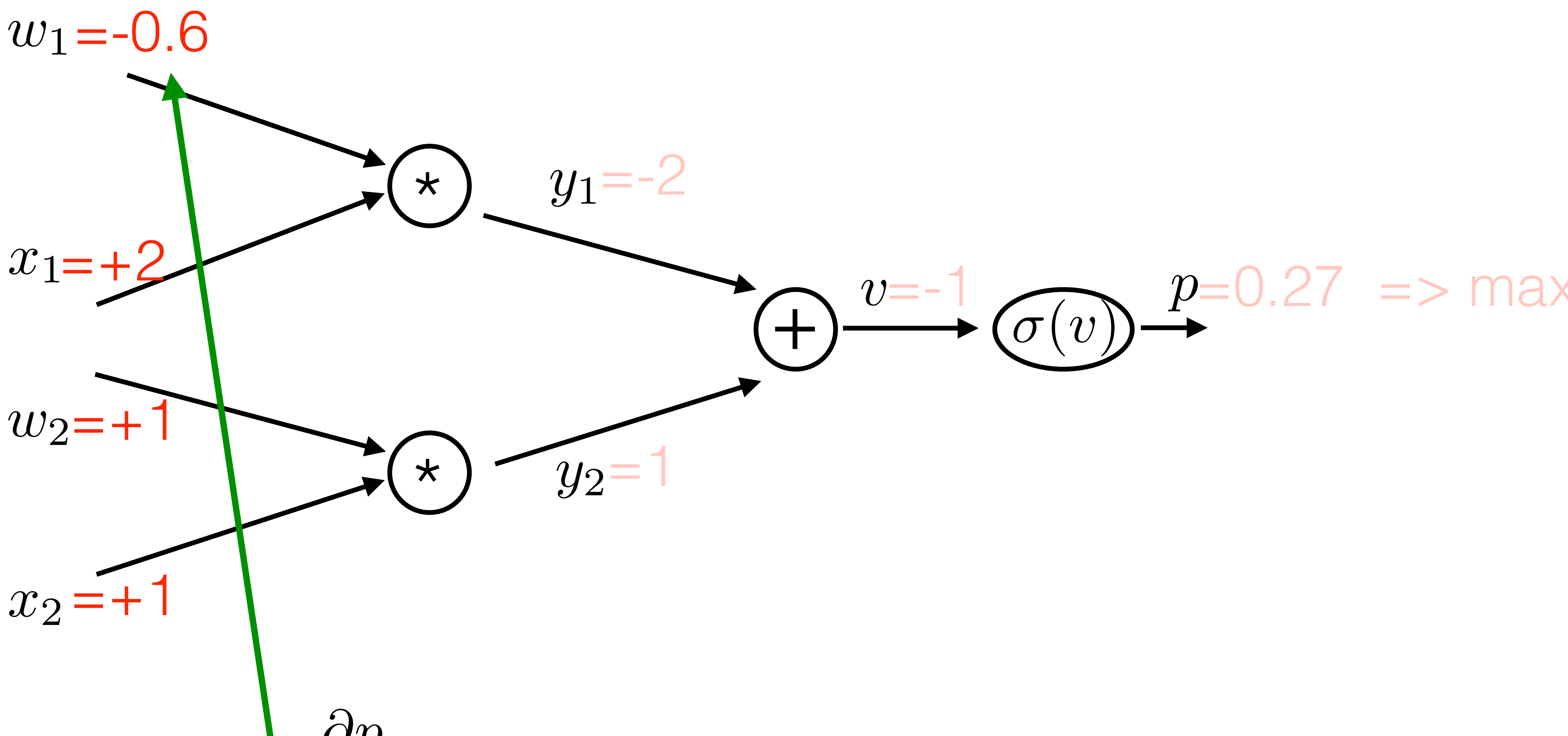
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$

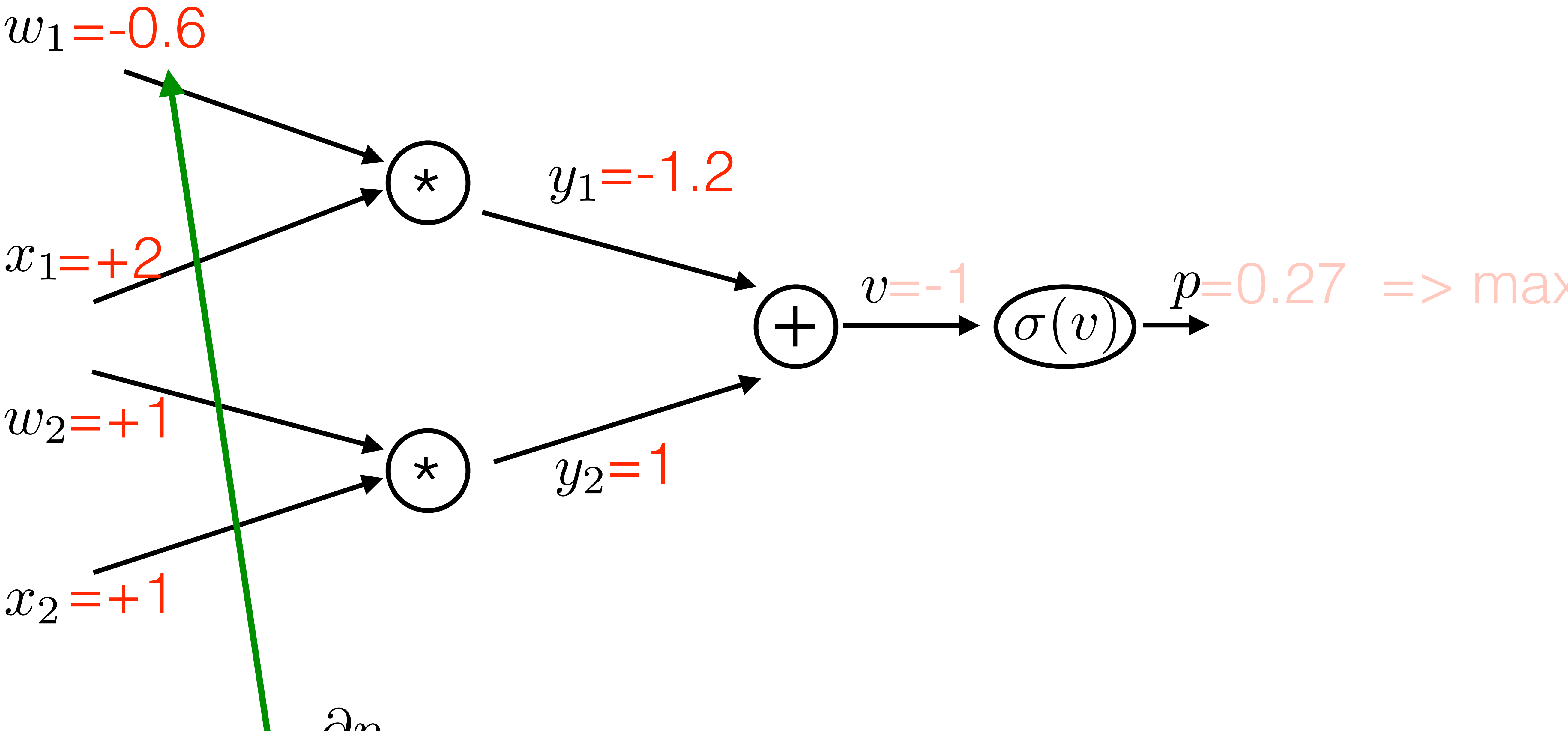
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $$\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$$

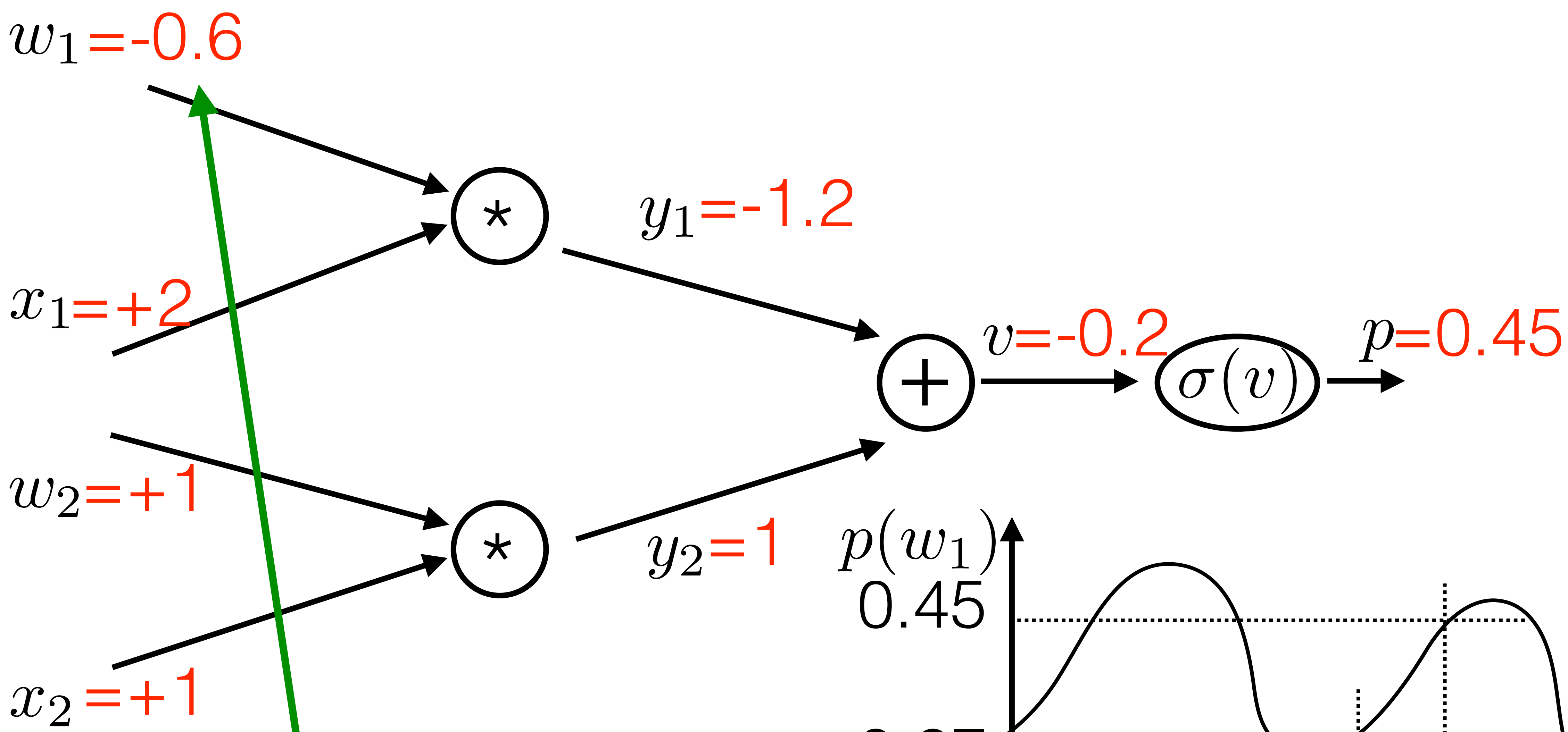
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p

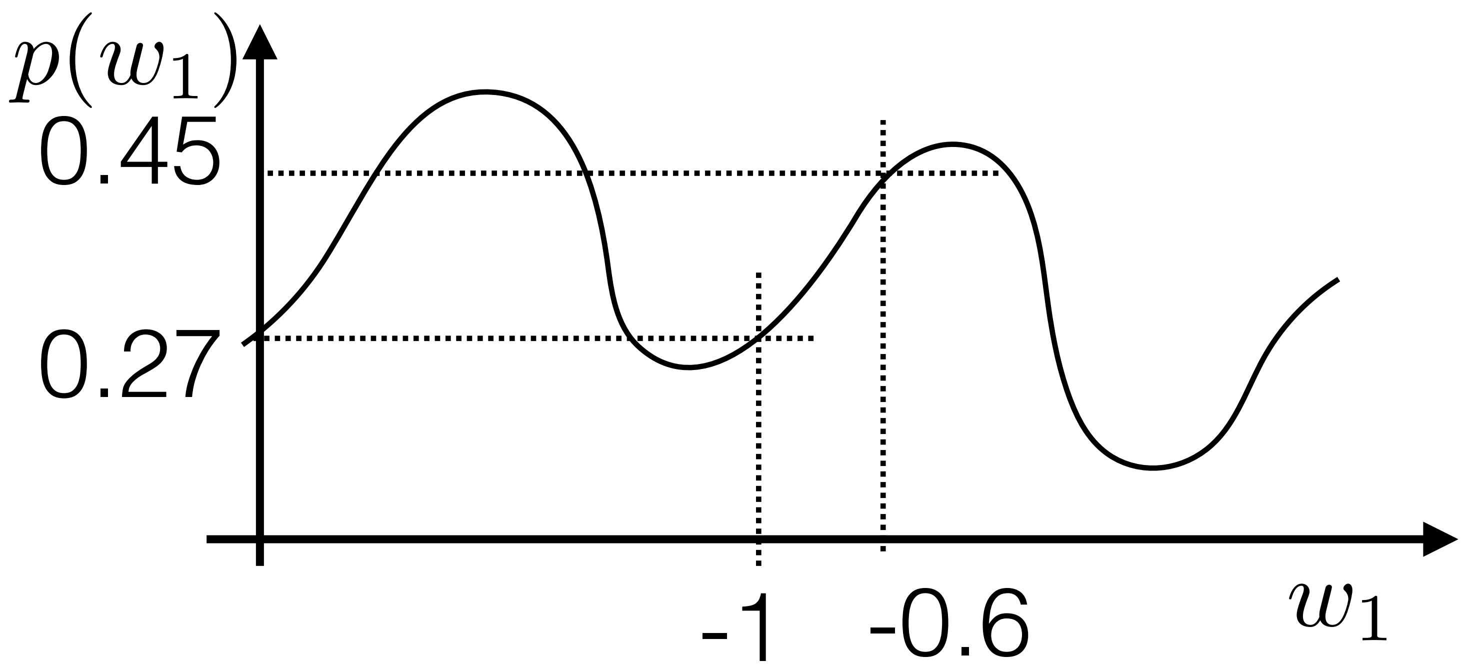


$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\frac{\partial p}{\partial w_1} = \frac{\partial p}{\partial v}\frac{\partial v}{\partial y_1}\frac{\partial y_1}{\partial w_1}$

Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p

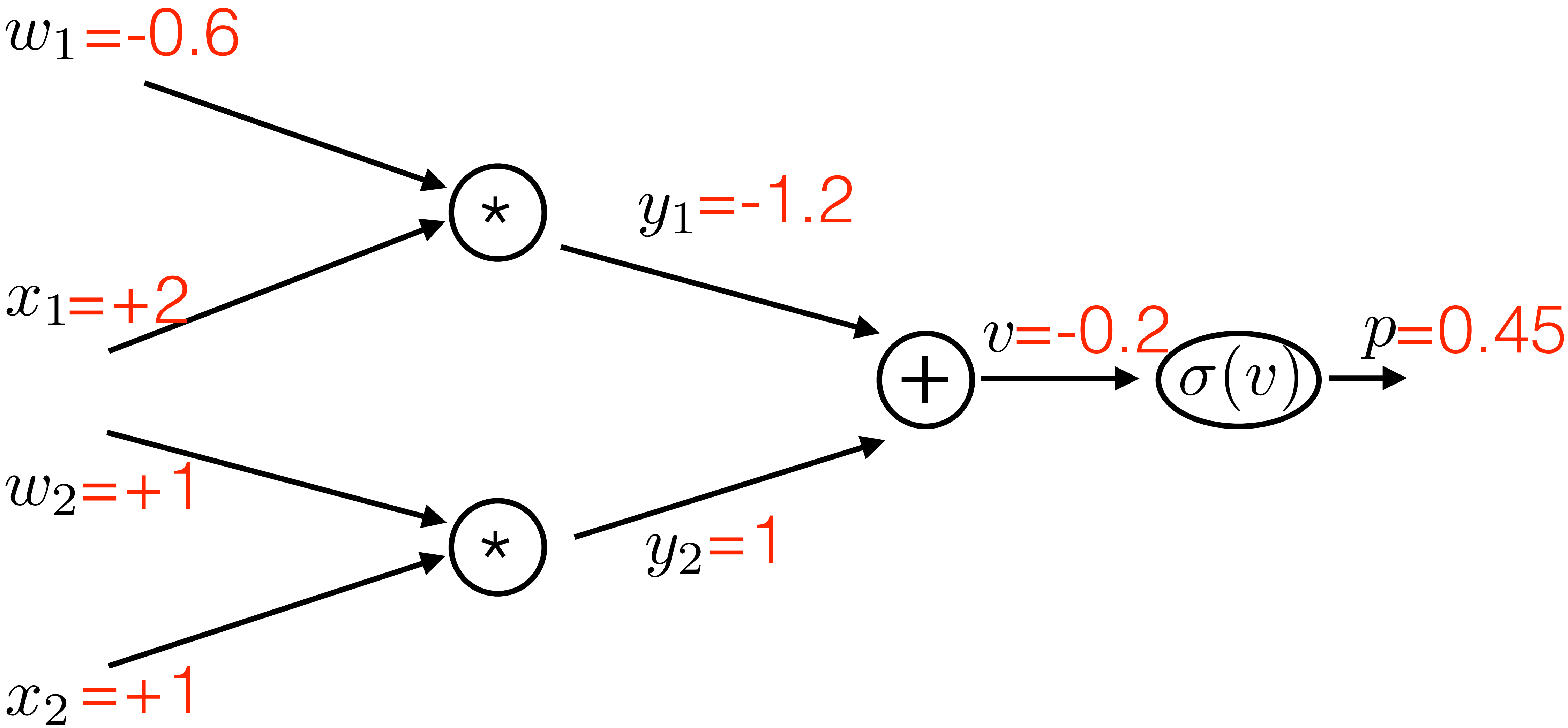




$$w_1 = w_1 + \frac{\partial p}{\partial w1}$$

Chain-rule in computational graph $\dfrac{\partial p}{\partial w_1} = \dfrac{\partial p}{\partial v}\dfrac{\partial v}{\partial y_1}\dfrac{\partial y_1}{\partial w_1}$

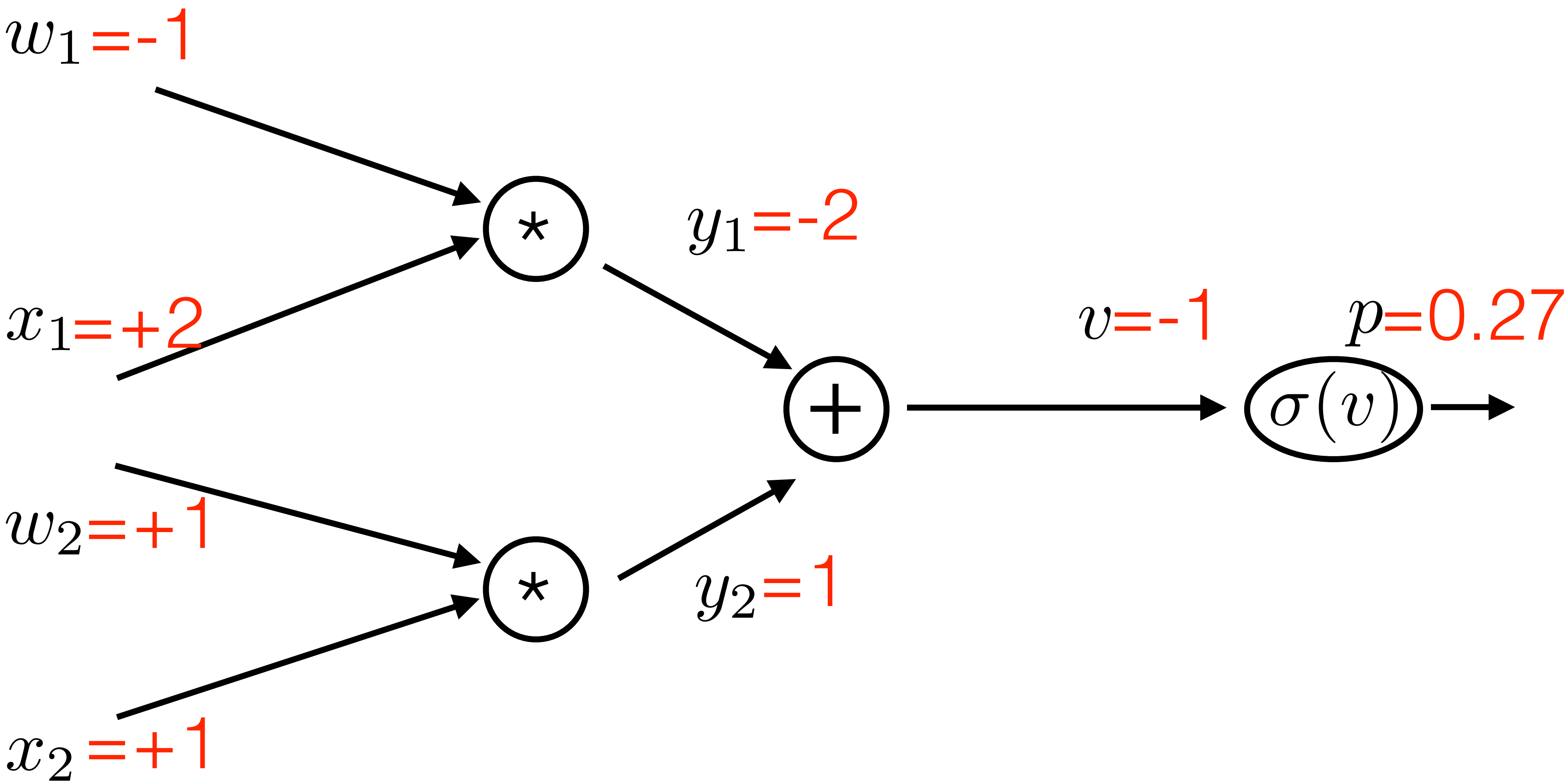
Example II: given x_1, x_2, w_1, w_2, change weight w_1 to increase p



Discussion:

- edge_gradient = upstream_gradient * local_gradient
- what is maximum p (bounds)? can I also update x_1?
- relation to learning (max p for positive samples)

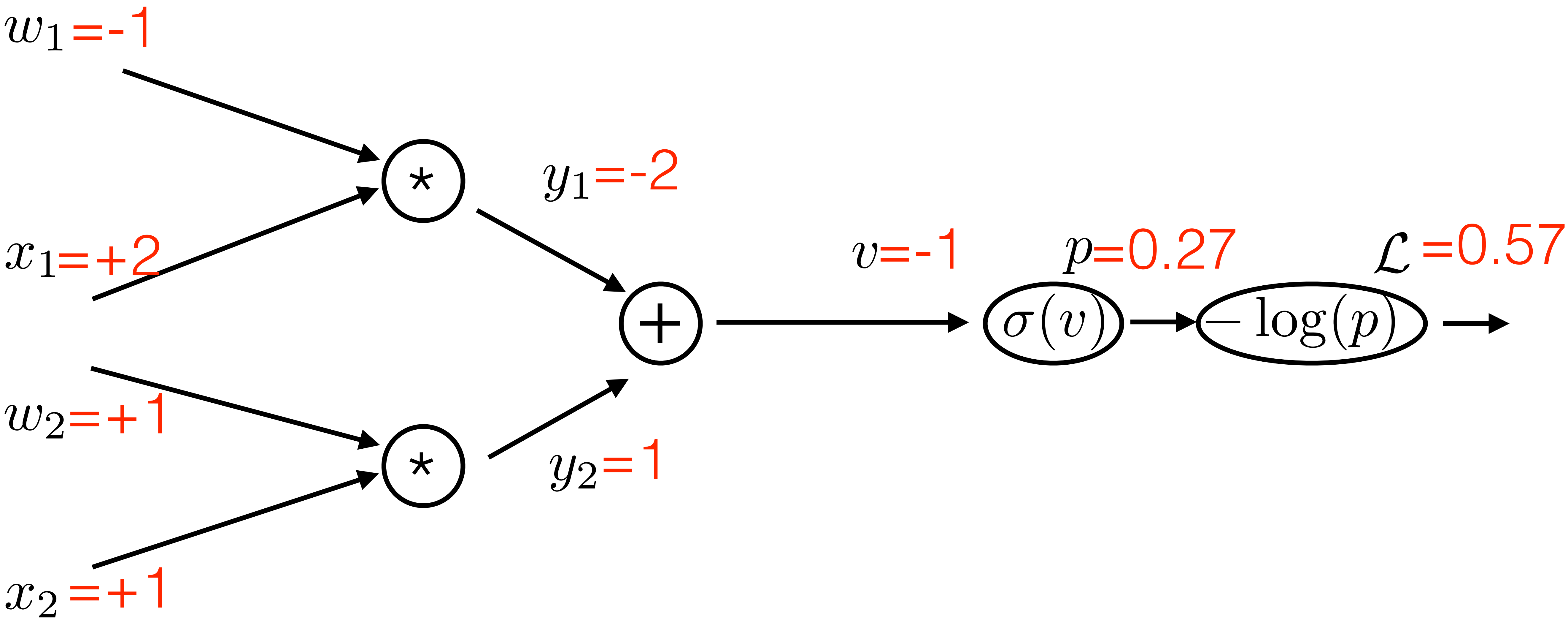
Computational graph of the learning- **adding loss layer**



MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

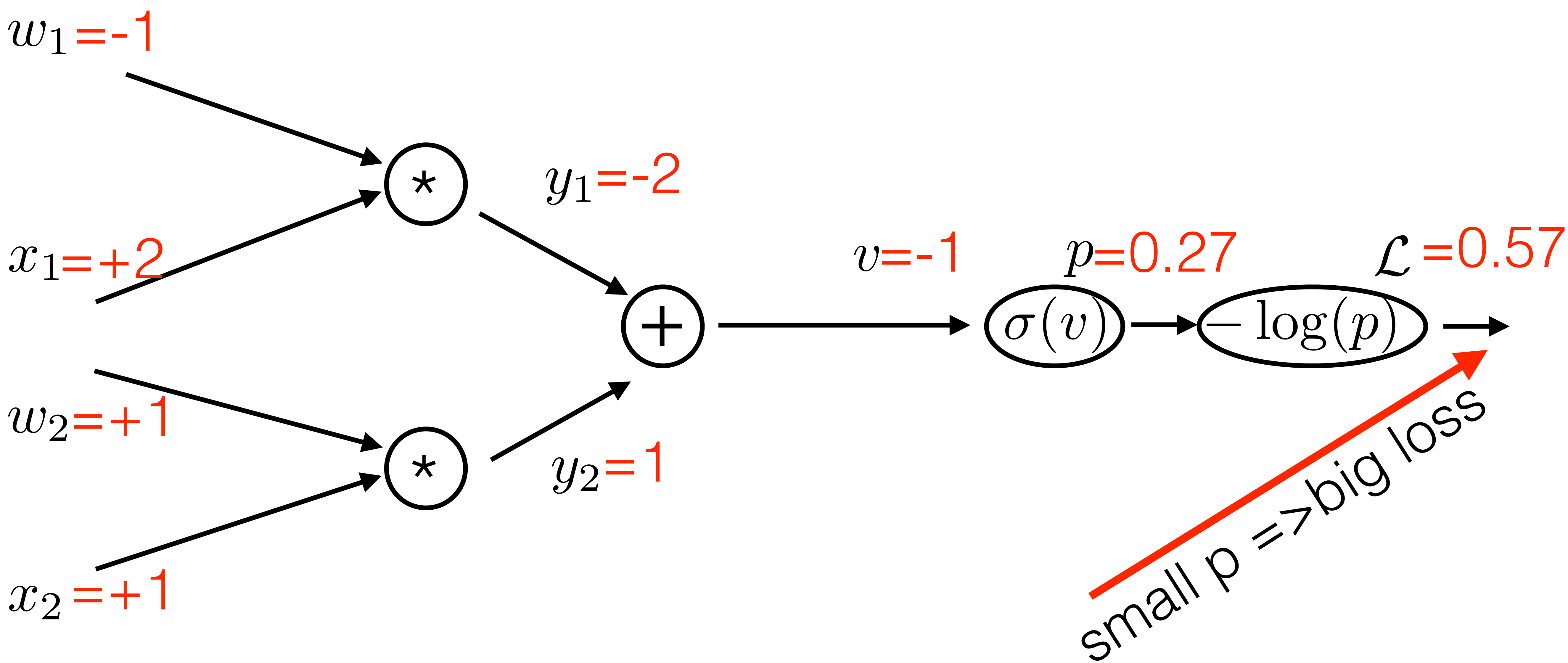
Computational graph of the learning from a **positive** sample



MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

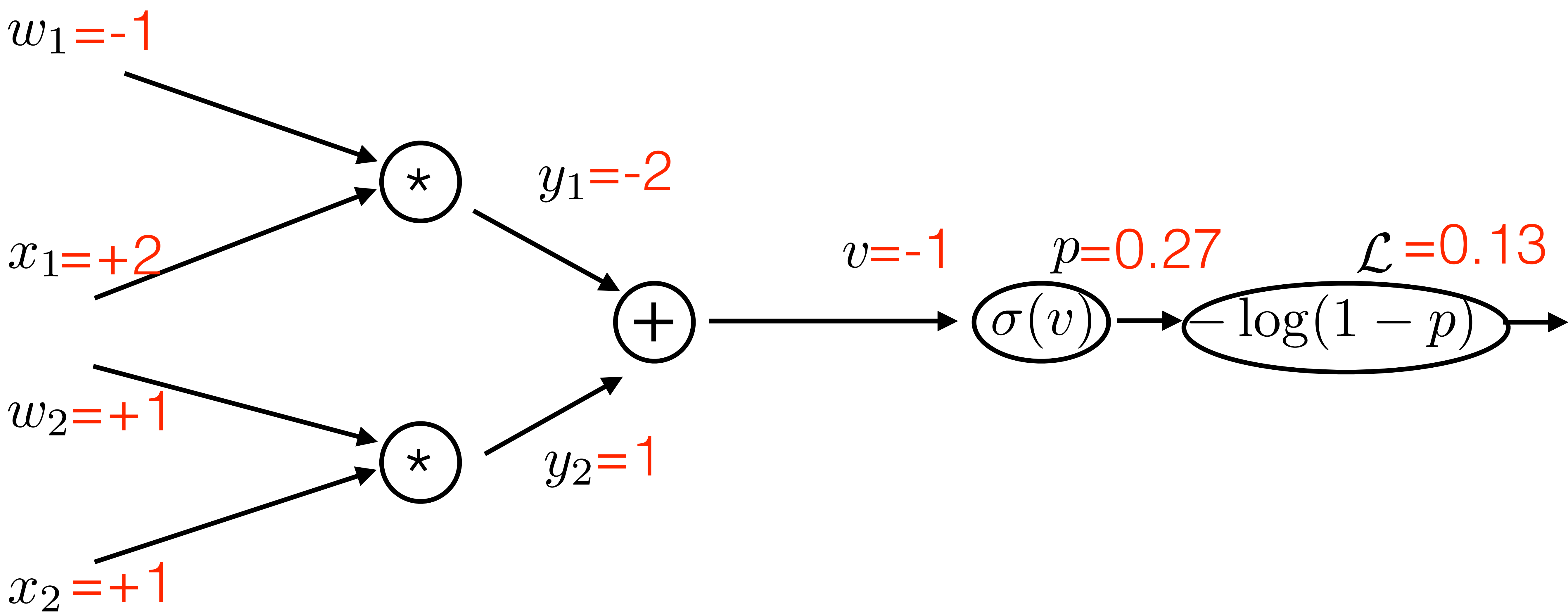
Computational graph of the learning from a **positive** sample



MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

Computational graph of the learning from a **negative** sample

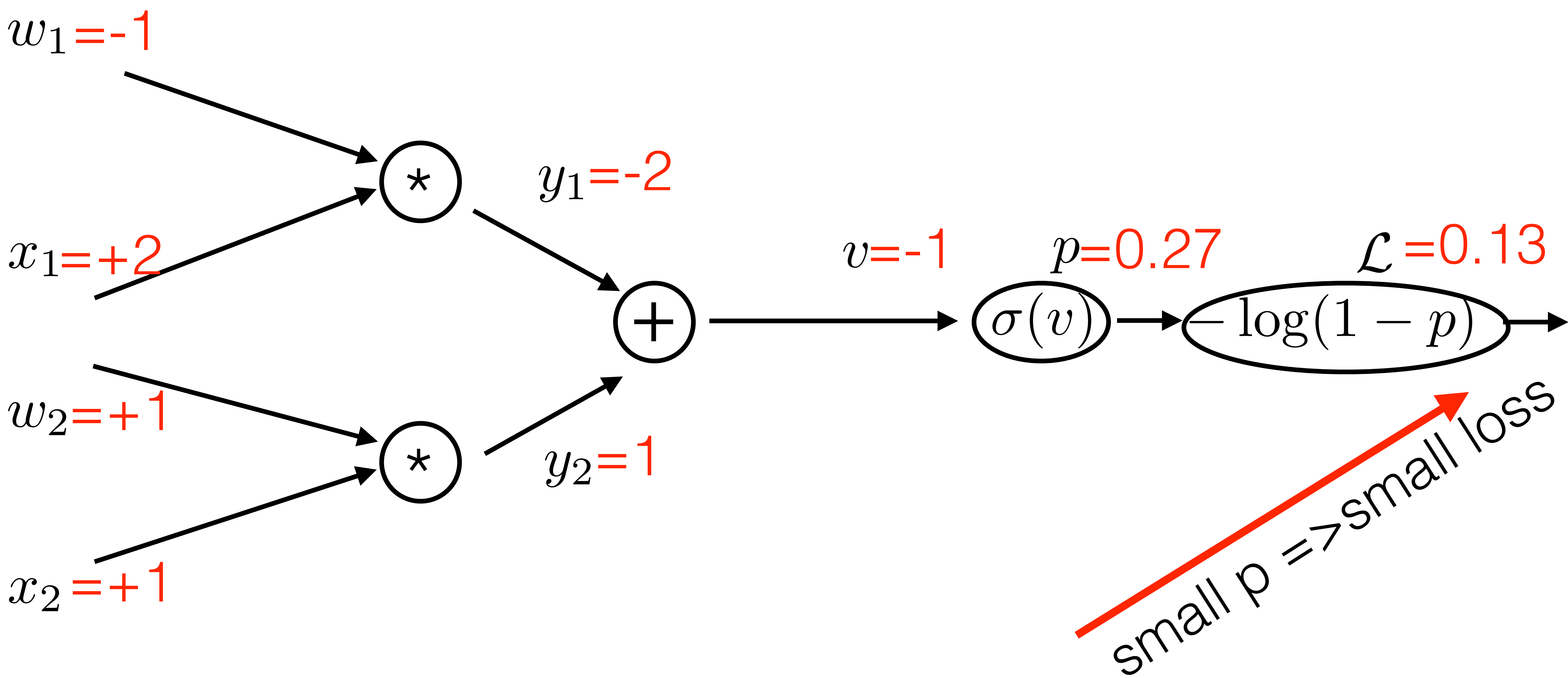


MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

Negative sample => p should be small => minimize $-\log(1-p)$

Computational graph of the learning from a **positive** sample

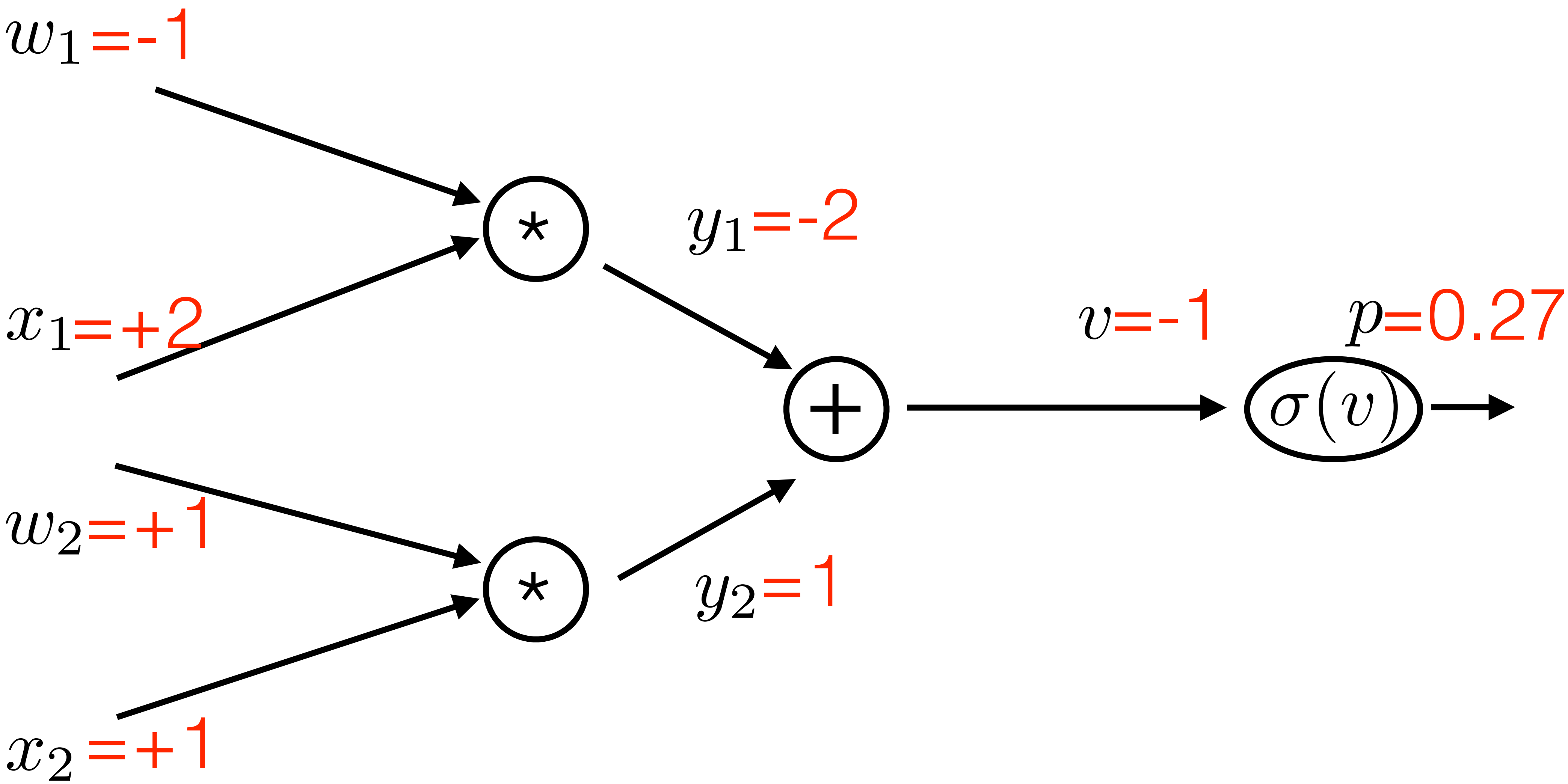


MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

Negative sample => p should be small => minimize $-\log(1-p)$

Computational graph of the learning- adding loss layer



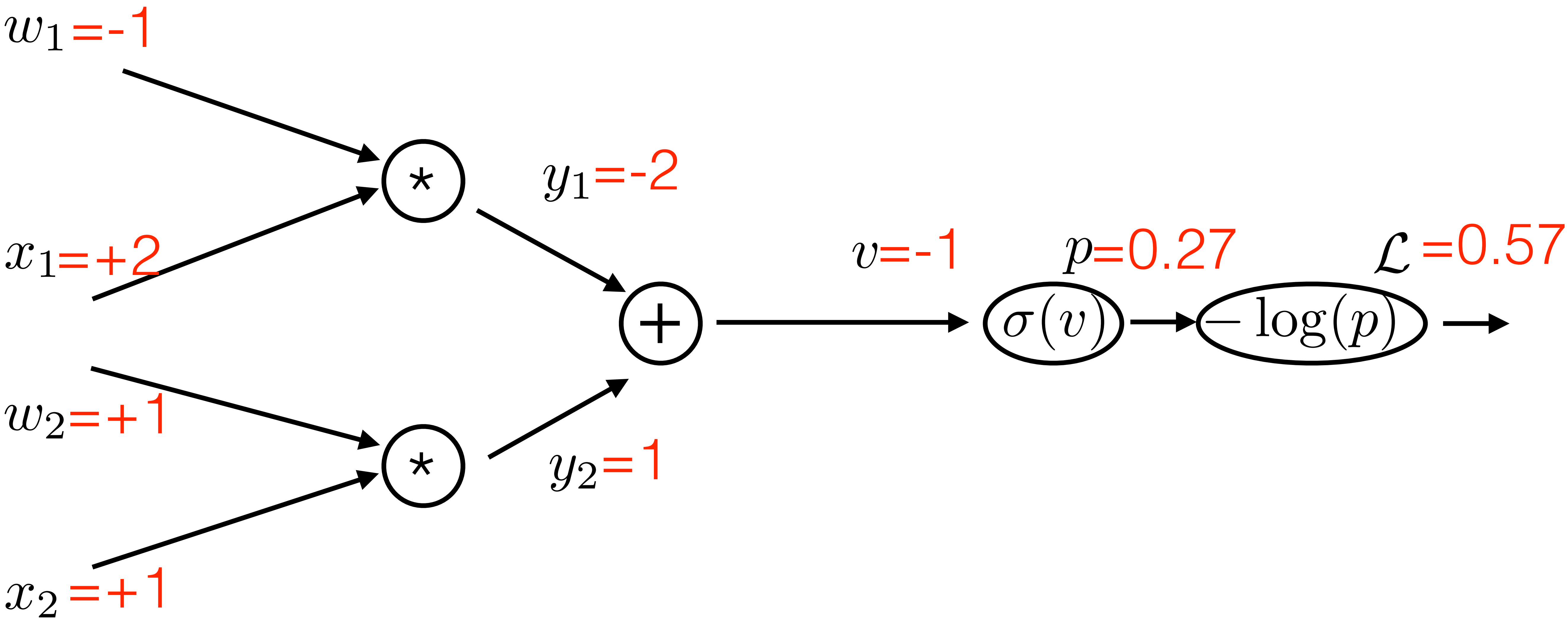
MAP estimate actually says:

Positive sample => p should be huge => minimize $-\log(p)$

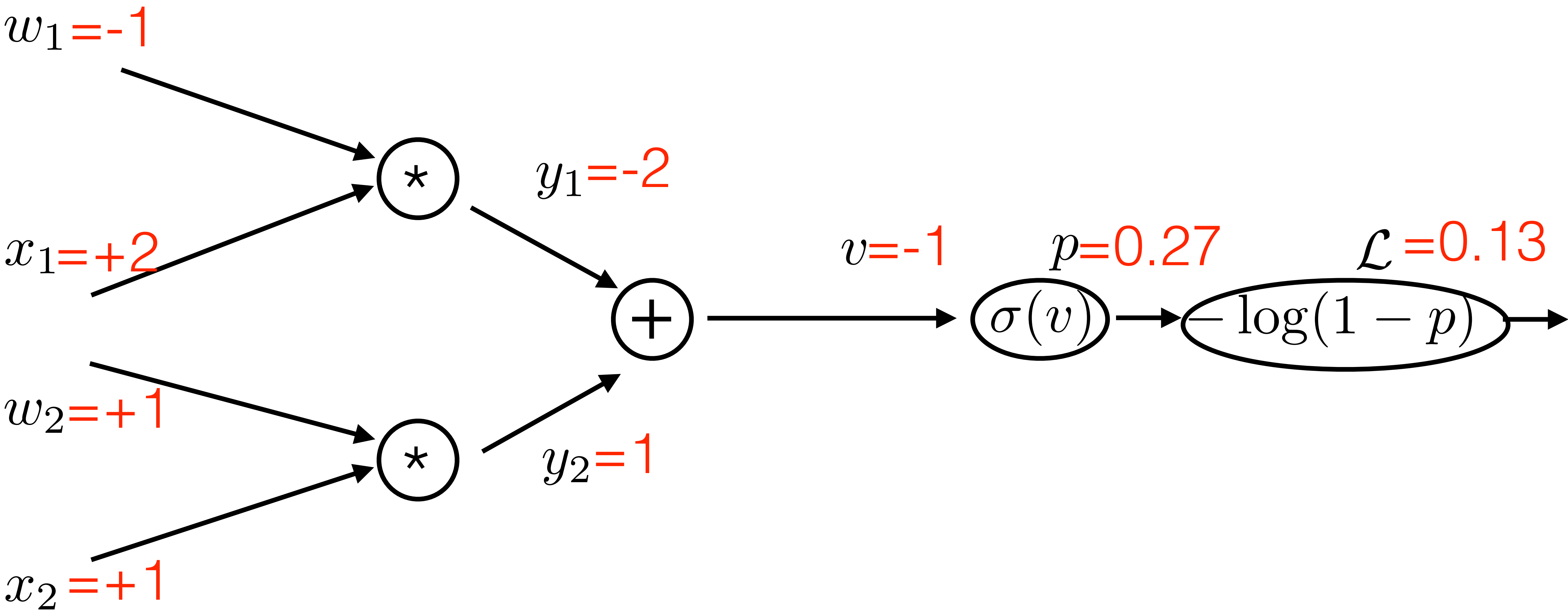
Negative sample => p should be small => minimize $-\log(1-p)$

We will unify computational graph for both cases as follows.

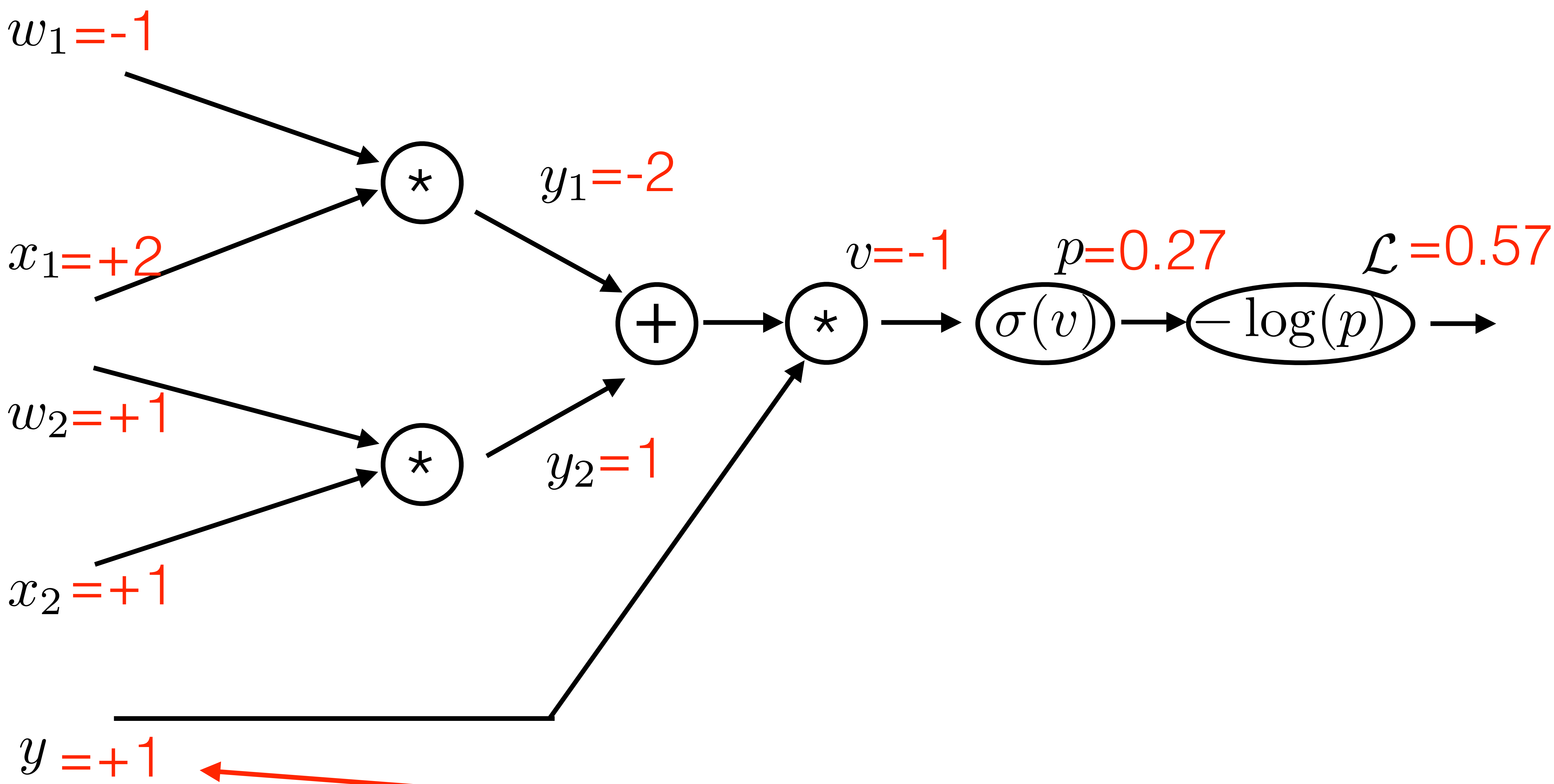
Computational graph of the learning from a **positive** sample



Computational graph of the learning from a **negative** sample

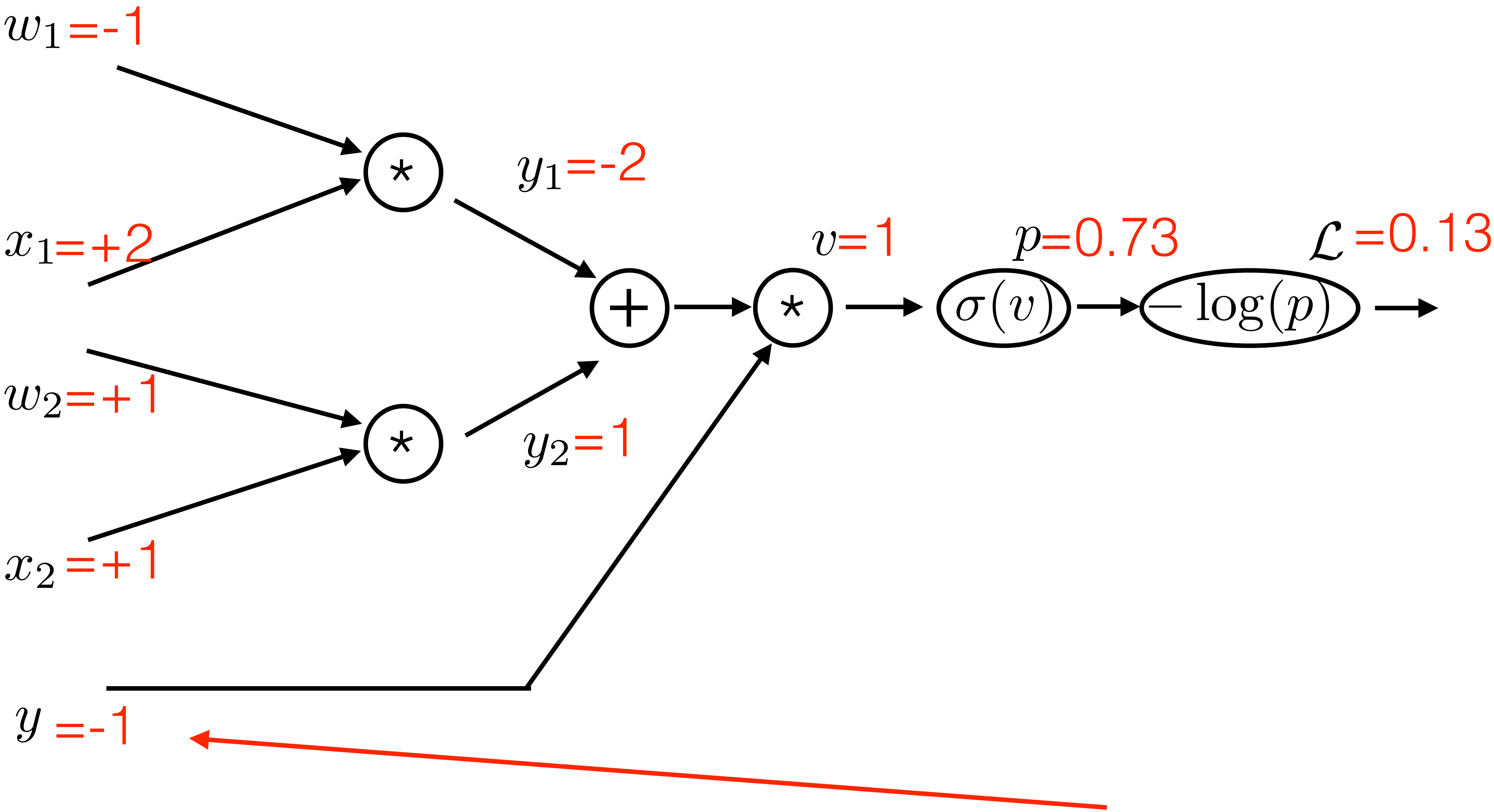


Computational graph of the learning



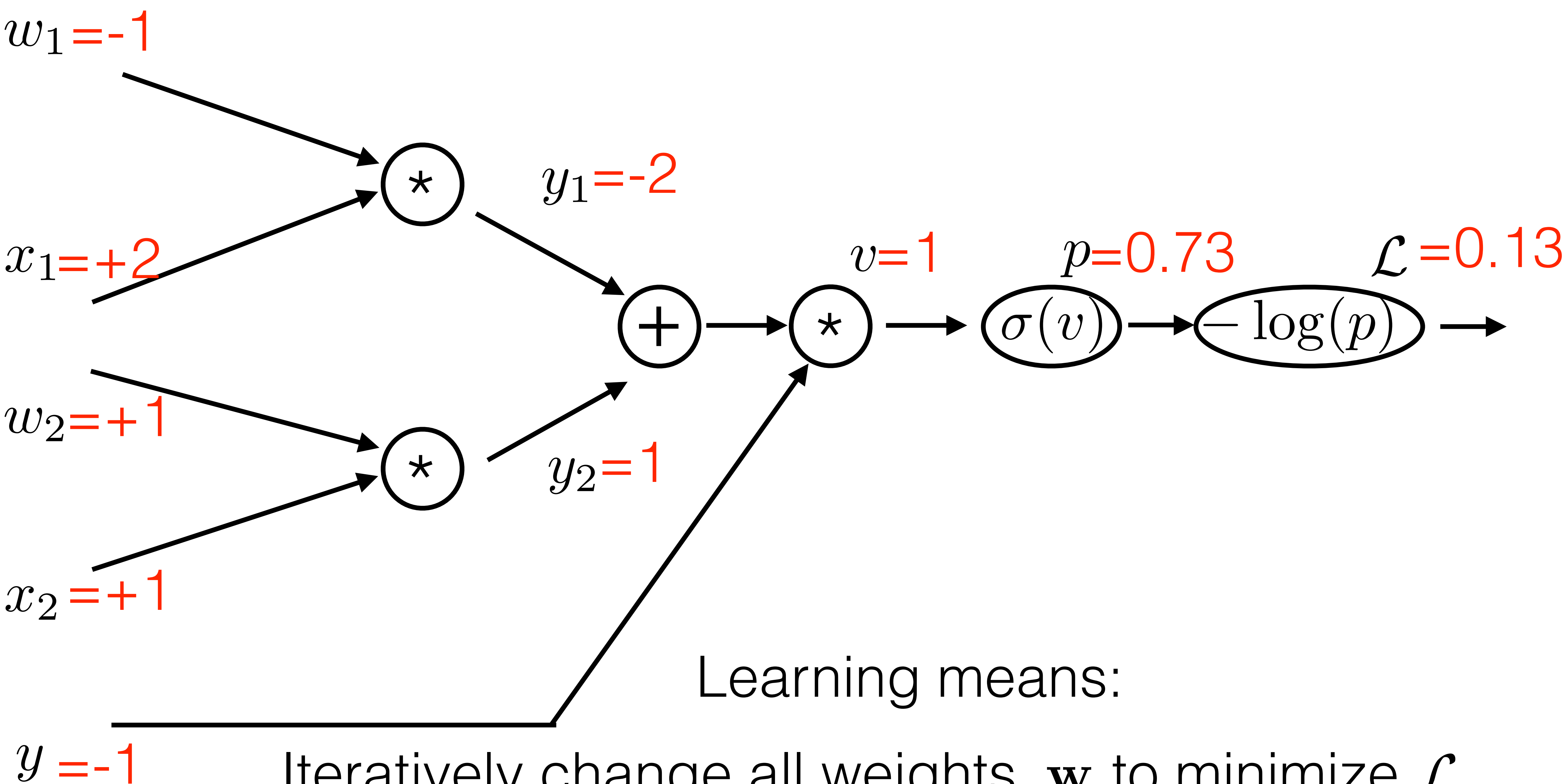
Computational graph for training on a **positive** sample

Computational graph of the learning



Computational graph for training on a **negative** sample

Computational graph of the learning

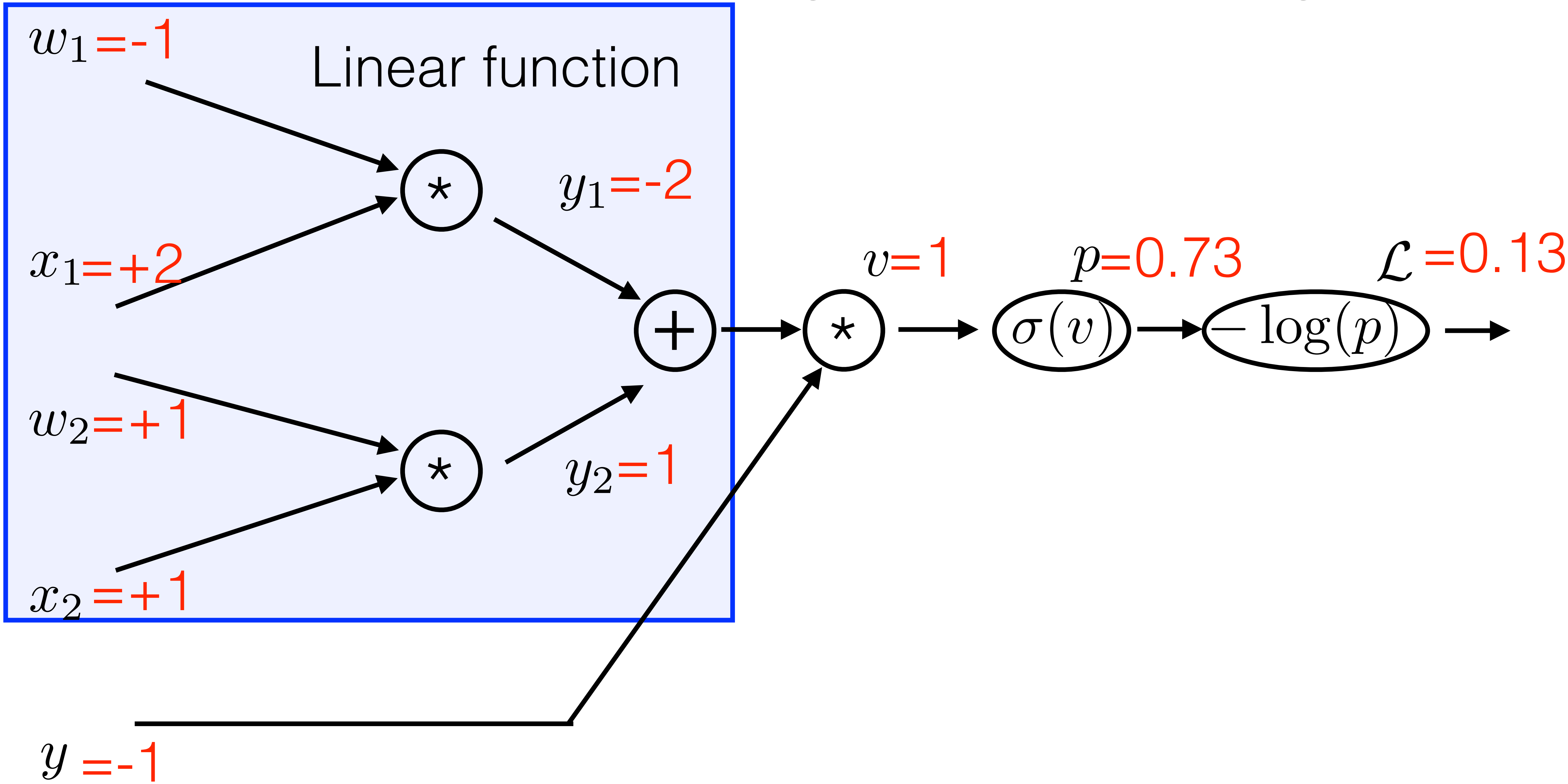


Learning means:

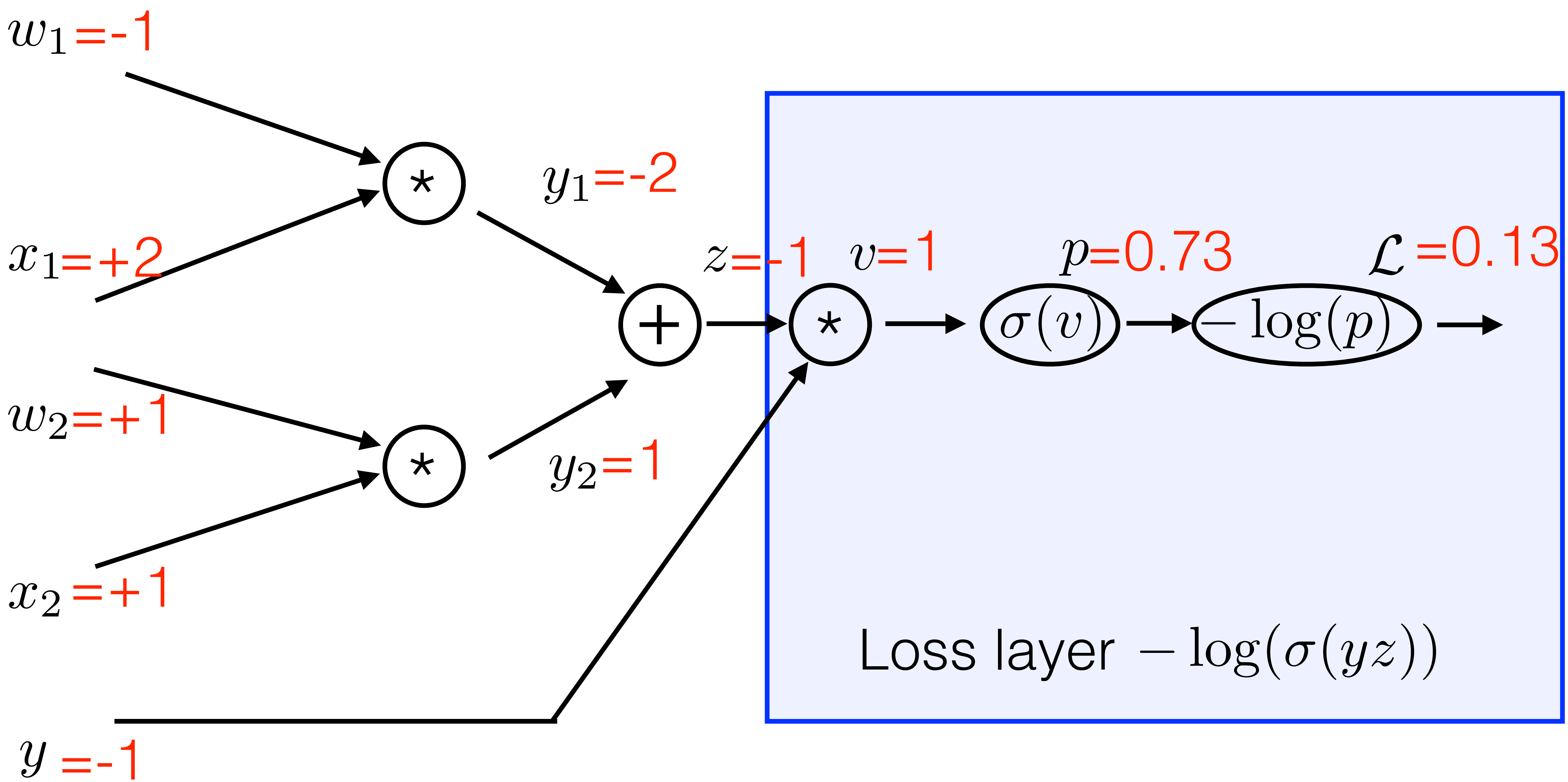
Iteratively change all weights $\mathbf{w}$ to minimize $\mathcal{L}$

$$\mathbf{w} = \mathbf{w} - \alpha \left[\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}\right]^\top \quad \text{where} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots\right]$$

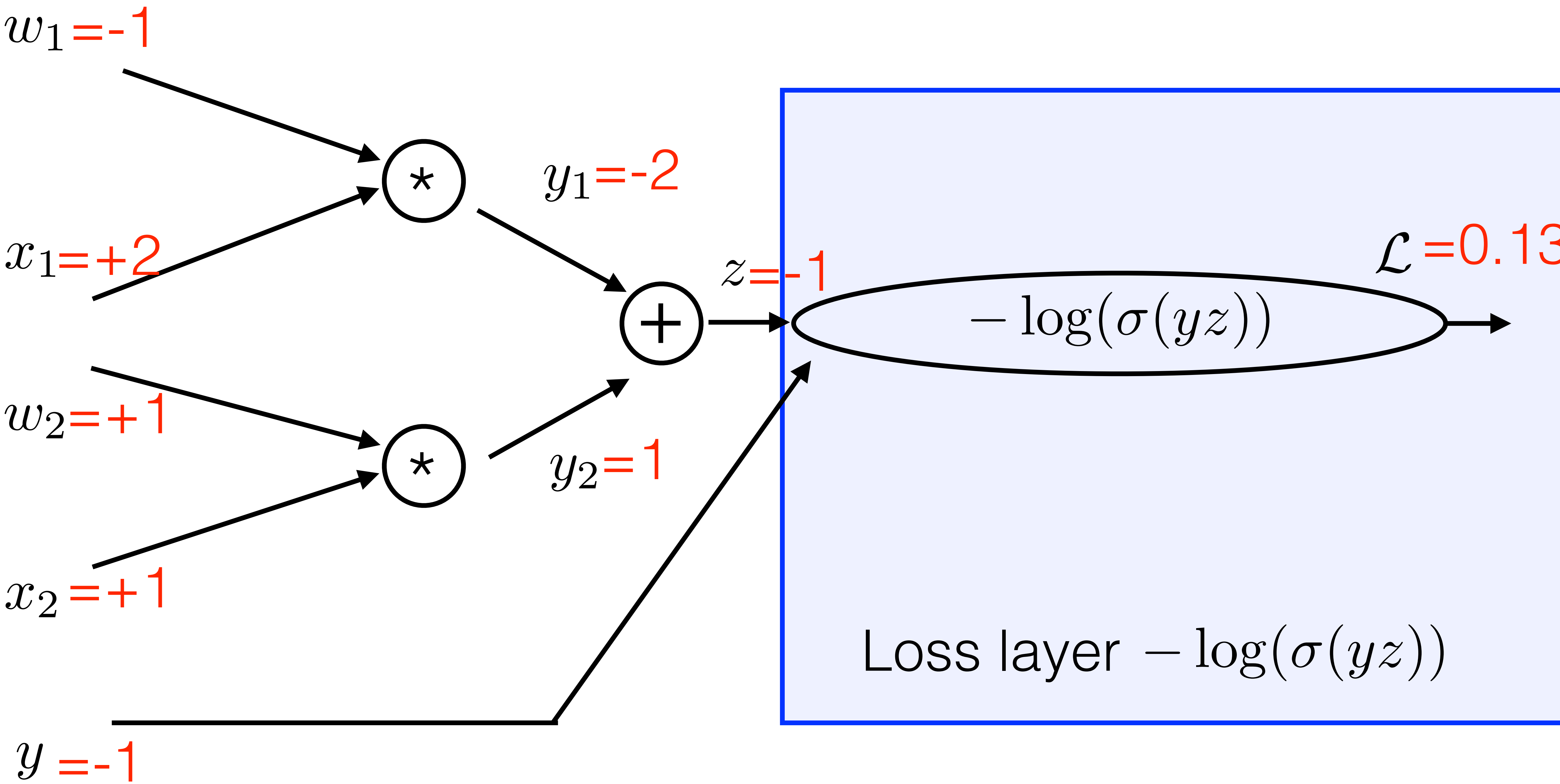
Computational graph of the learning



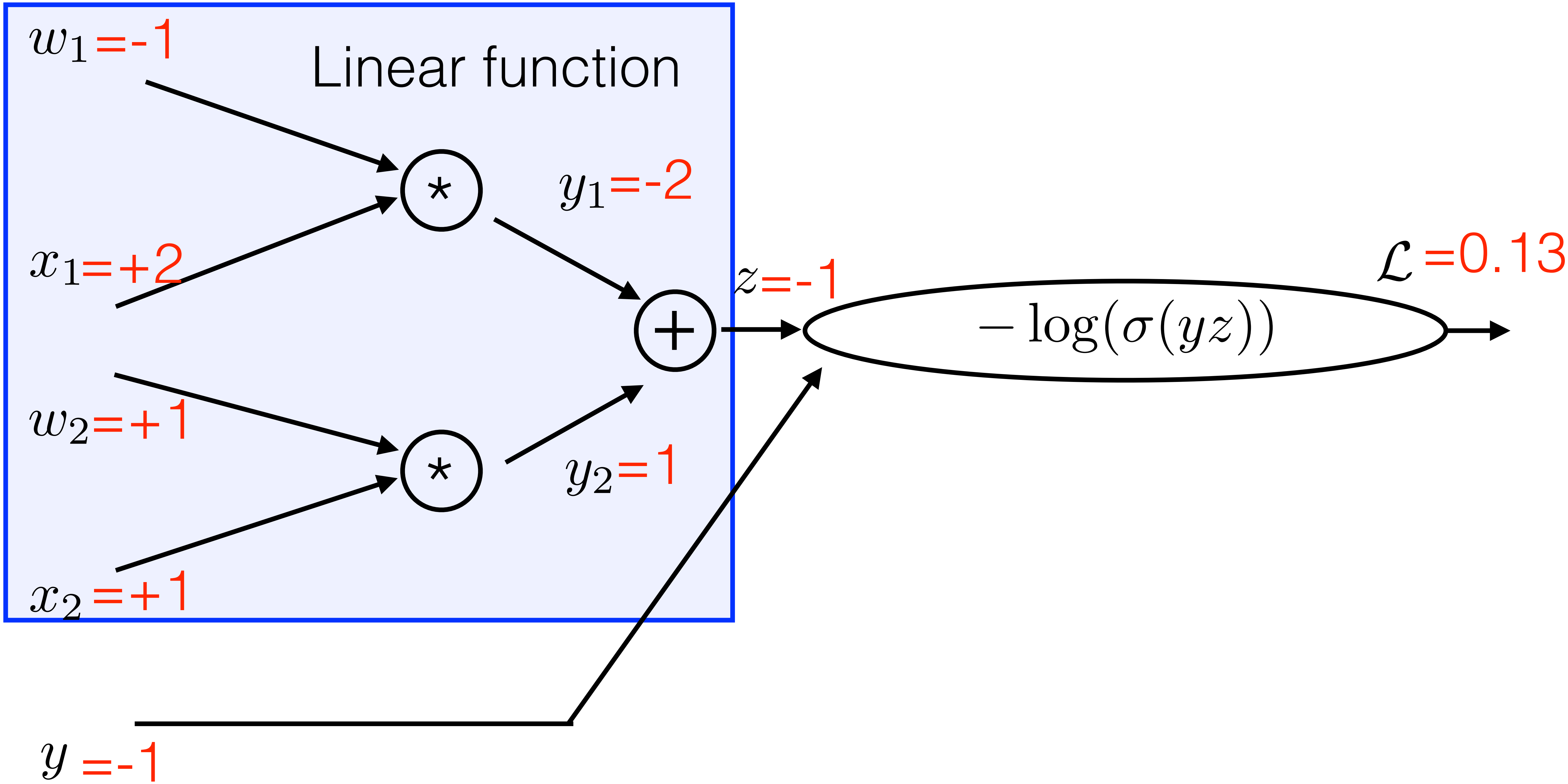
Computational graph of the learning



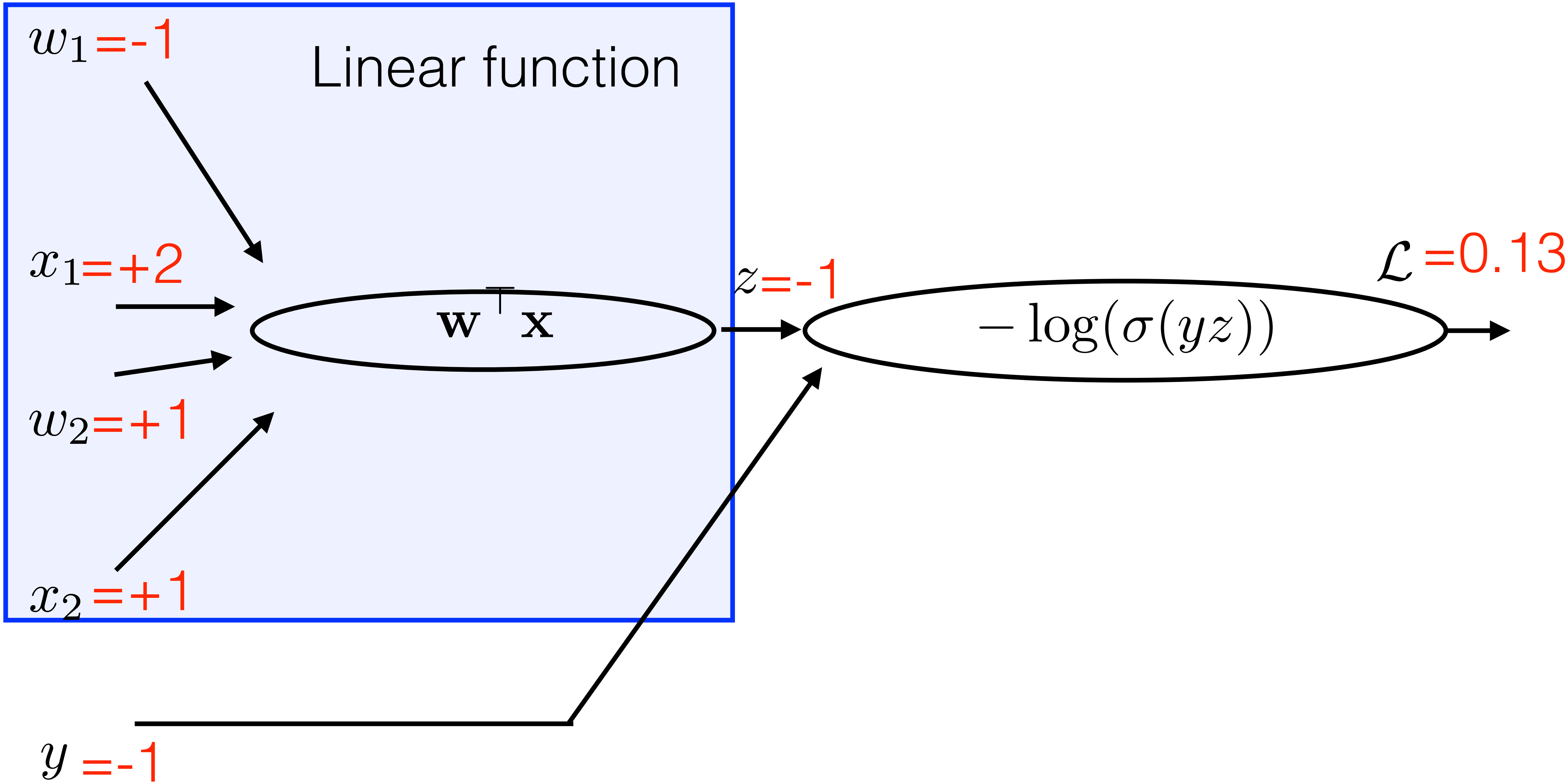
Backprop in vector representation



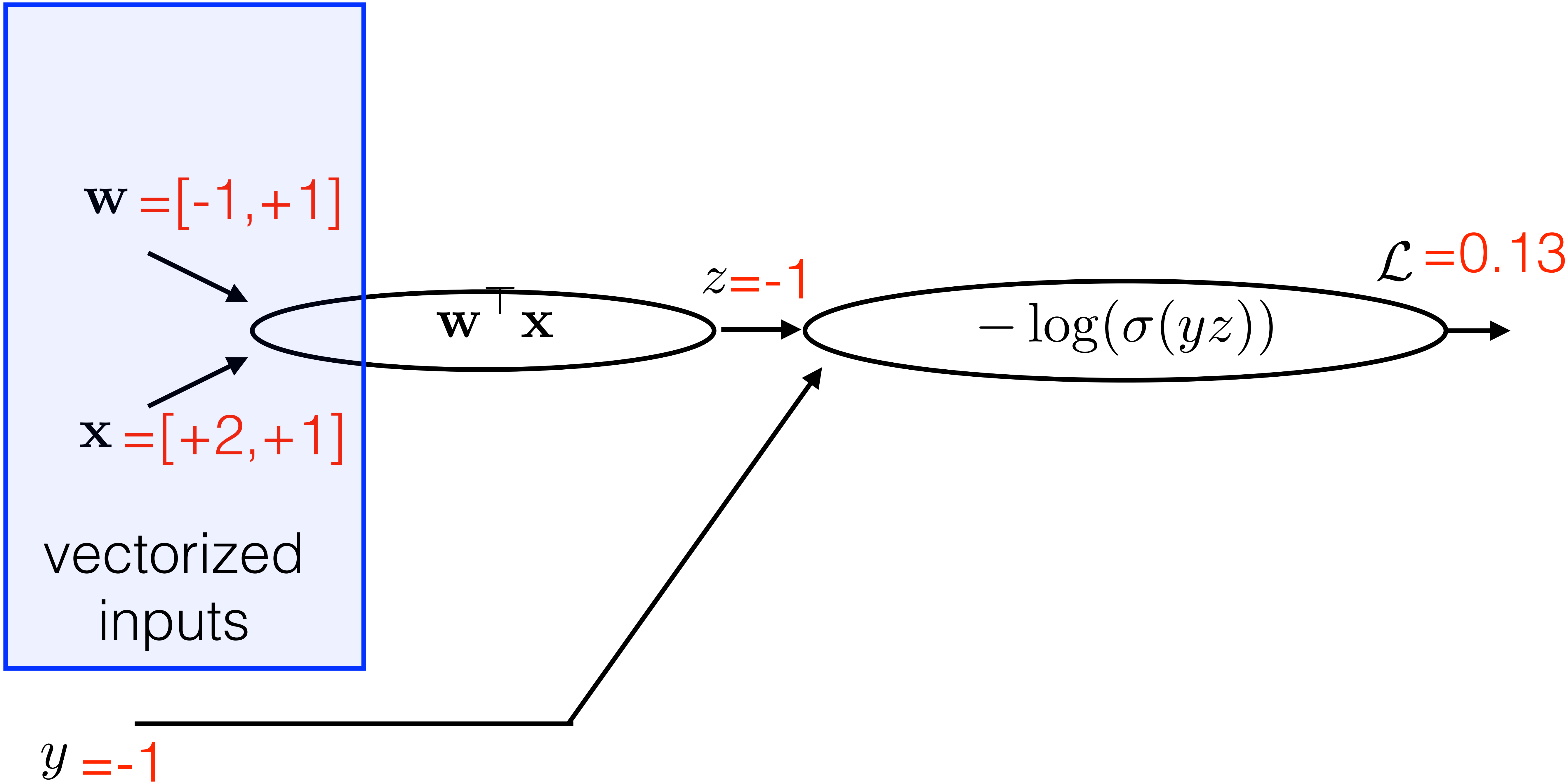
Backprop in vector representation



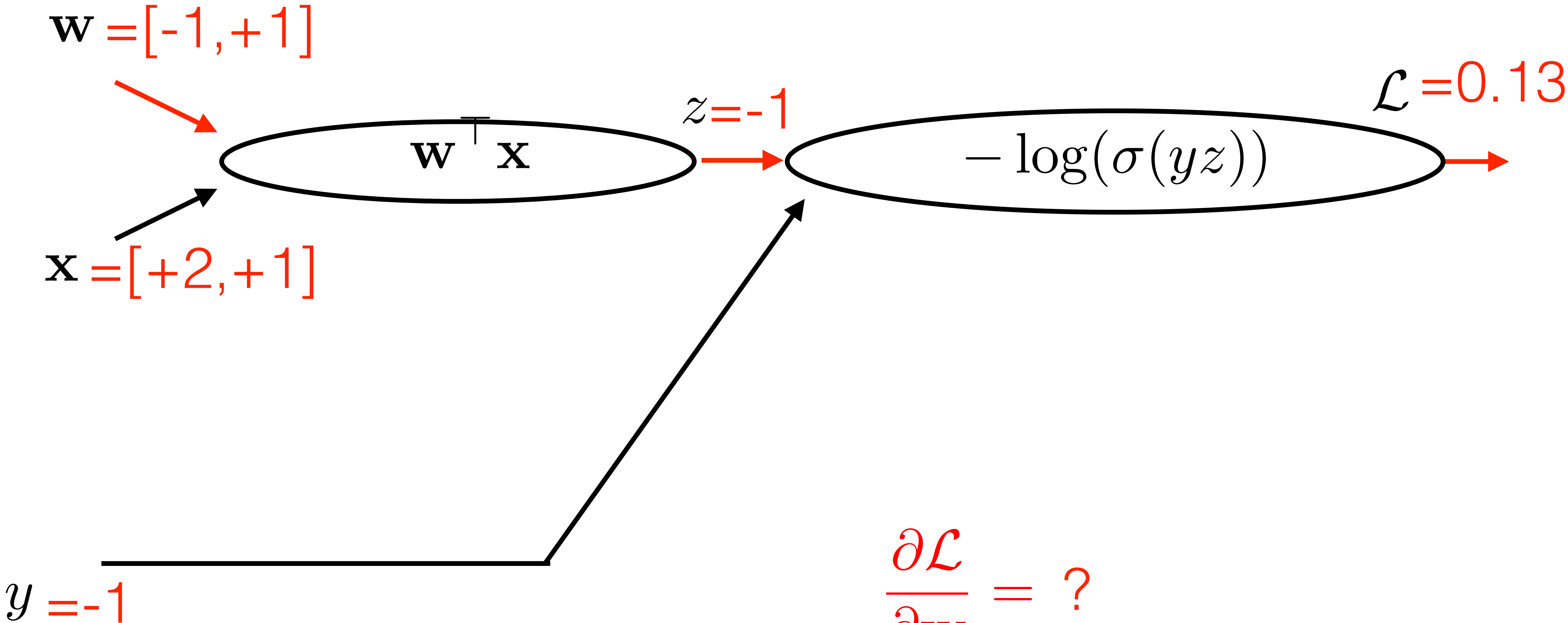
Backprop in vector representation



Backprop in vector representation

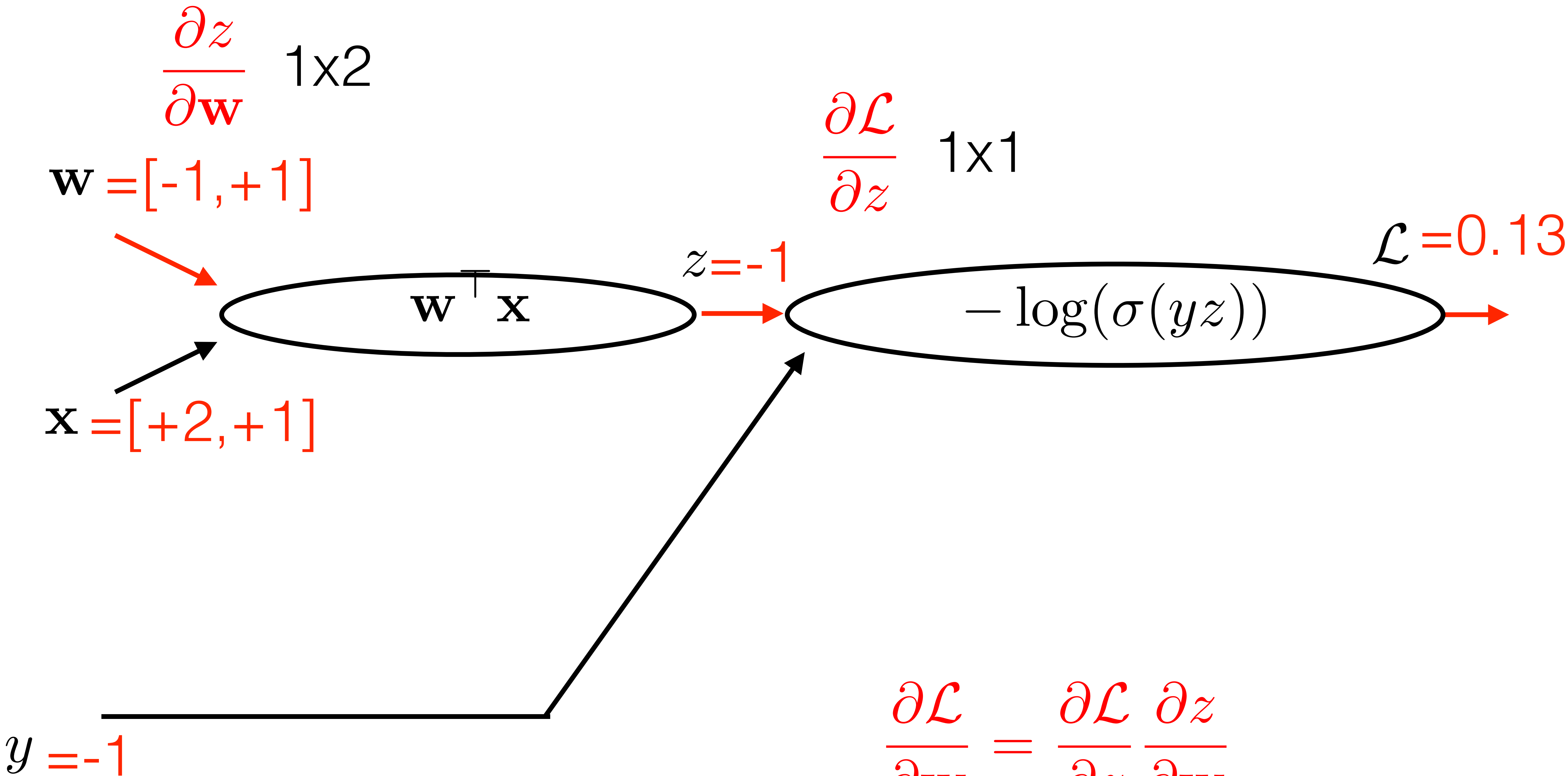


Backprop in vector representation



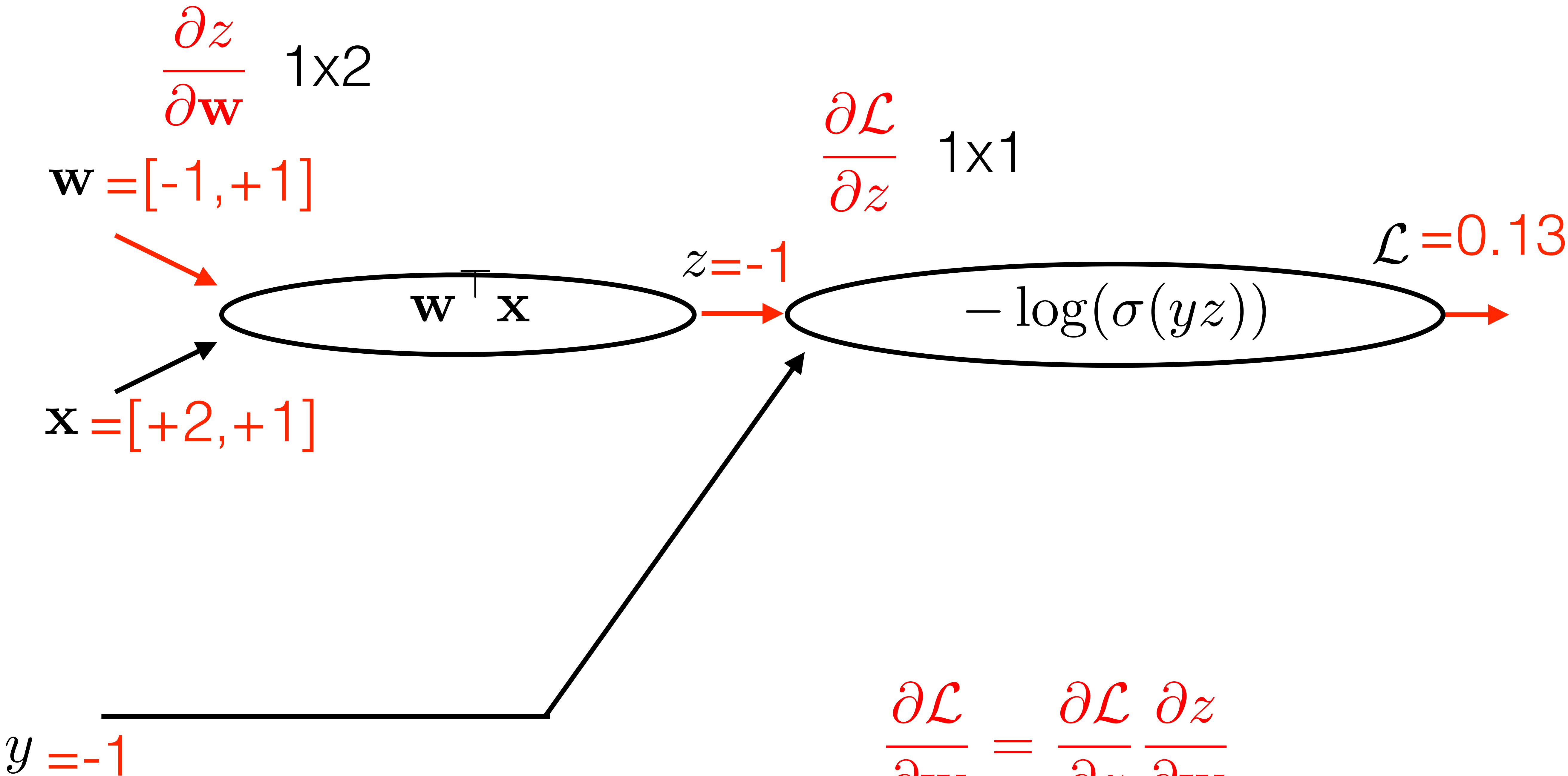
$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = ?$$

Backprop in vector representation



$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial \mathbf{w}}$$

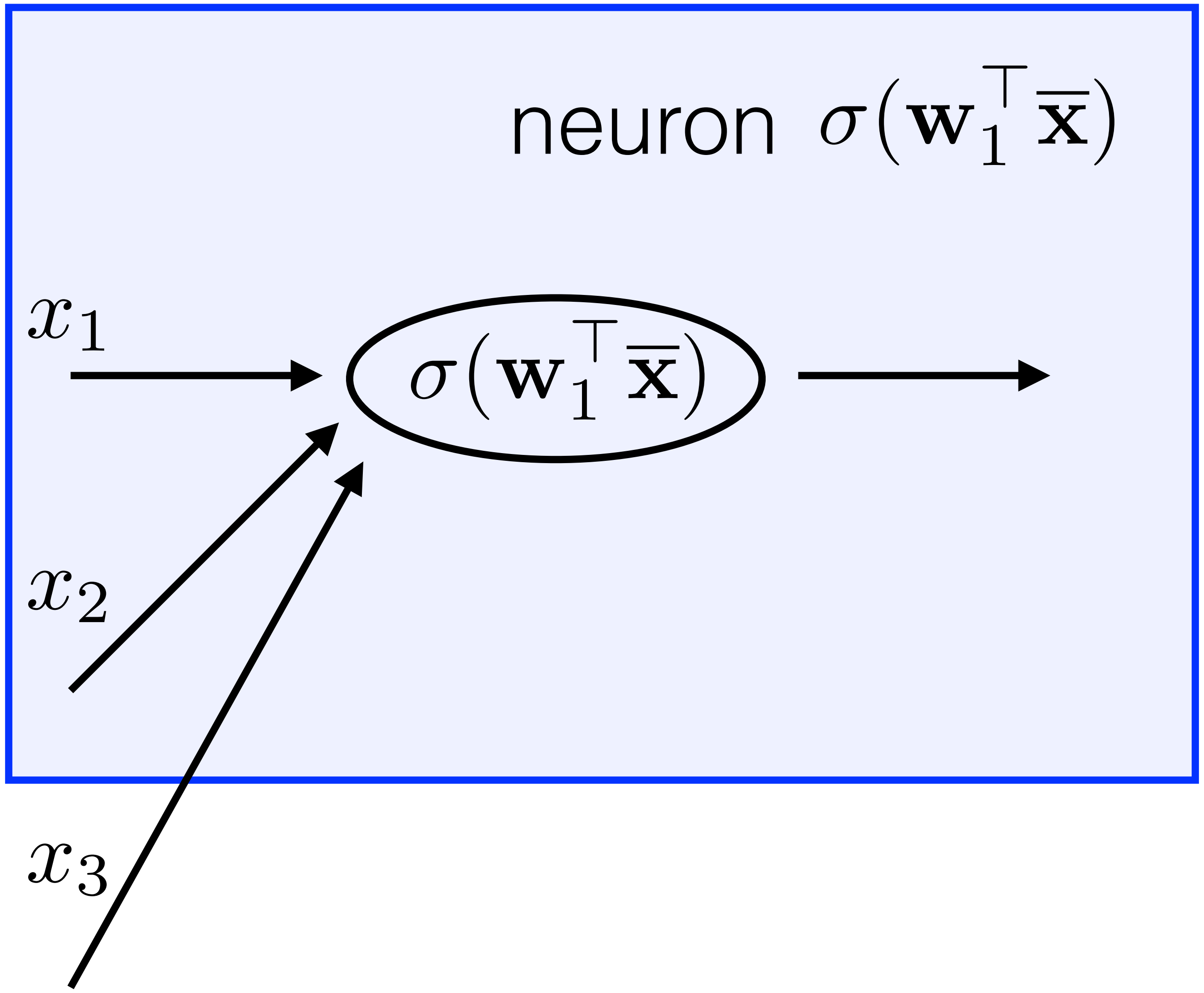
Backprop in vector representation



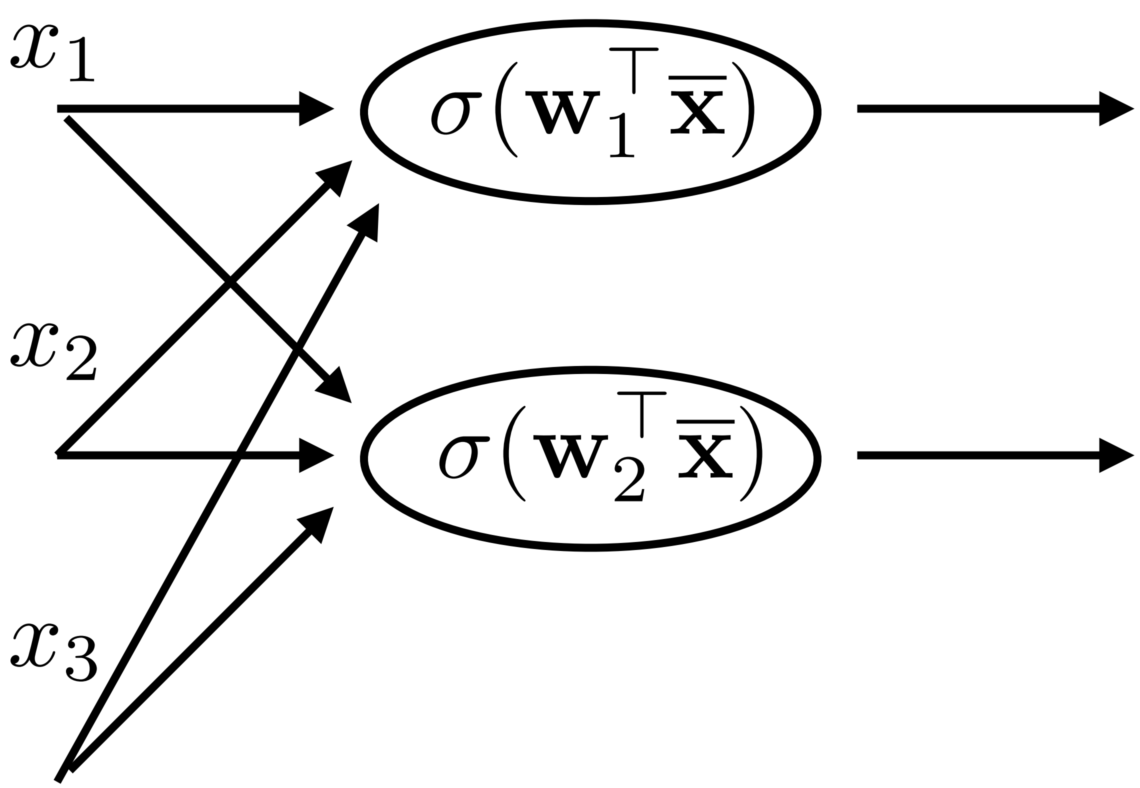
$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial \mathbf{w}}$$

Learning from multiple training samples means summing up the gradient over all samples

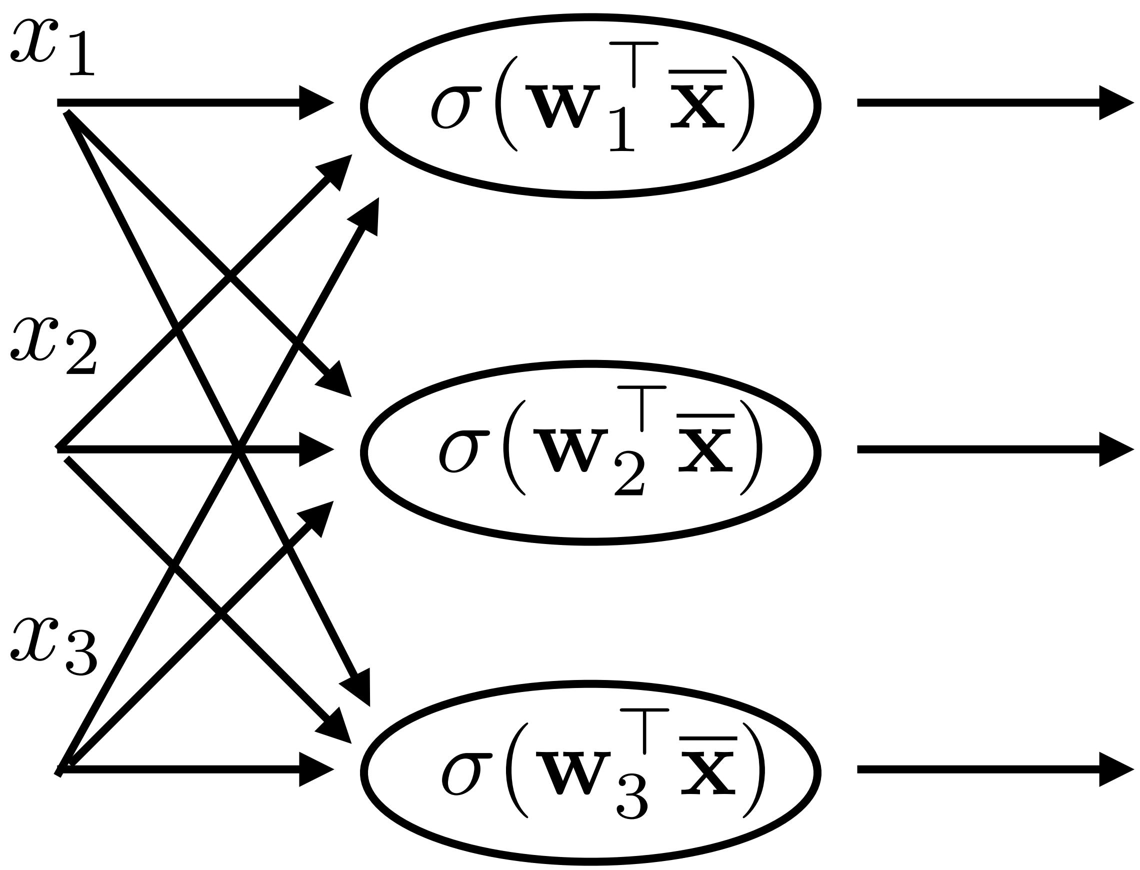
Multiple neurons and layers



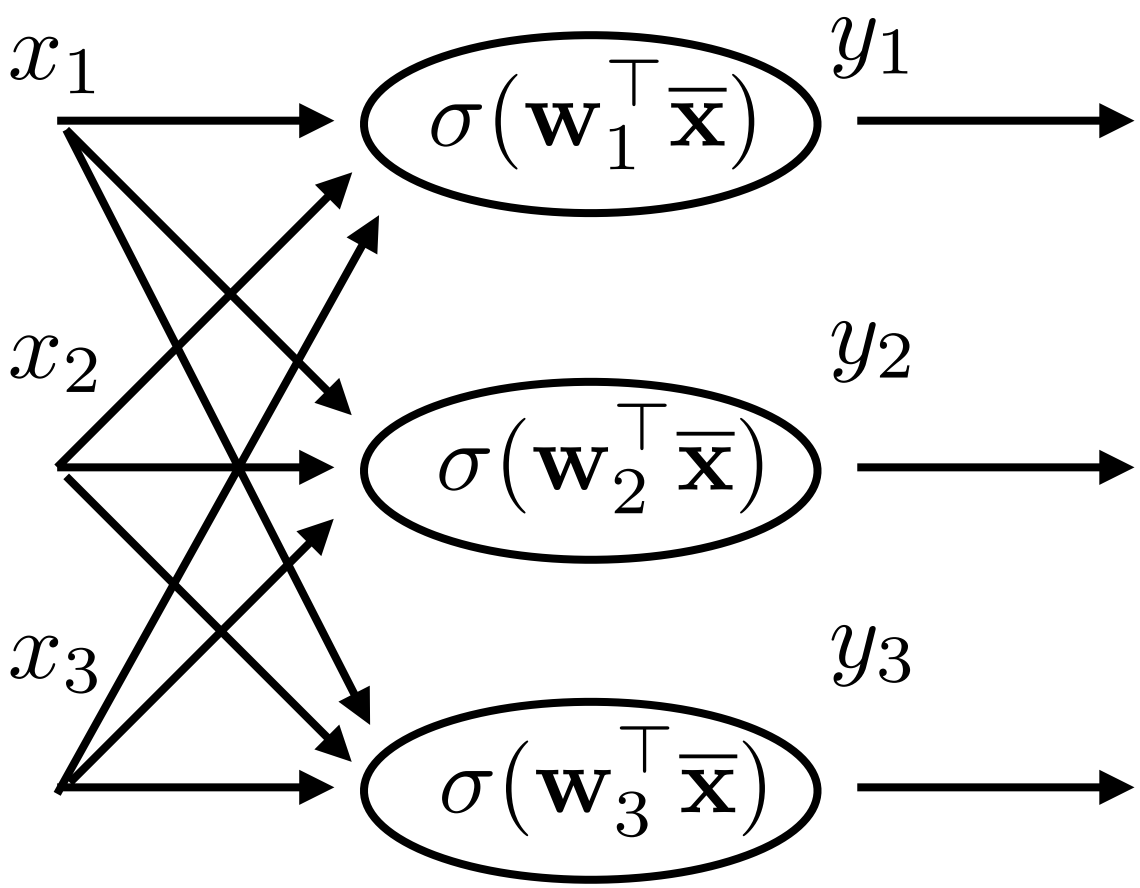
Multiple neurons and layers



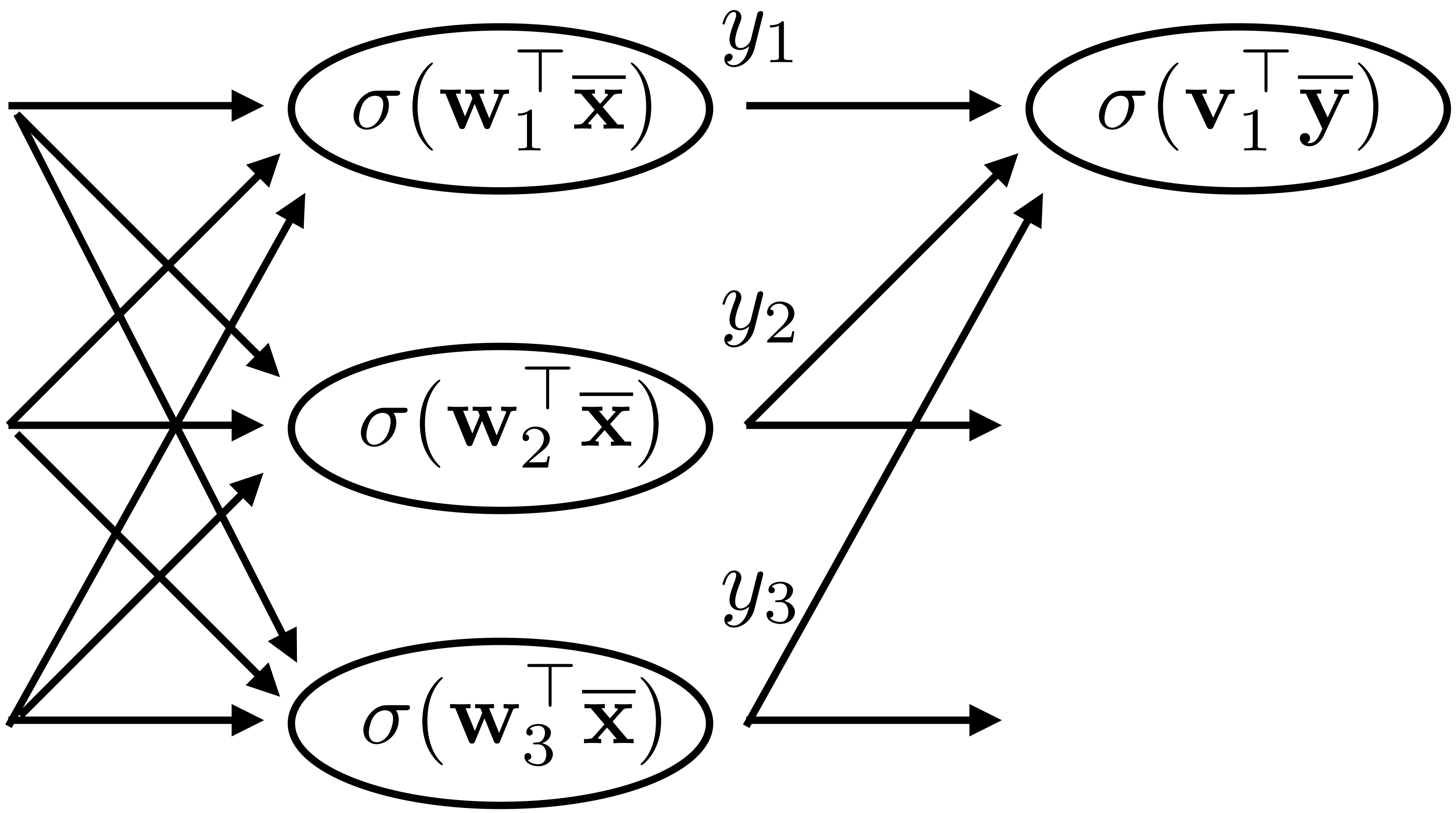
Multiple neurons and layers



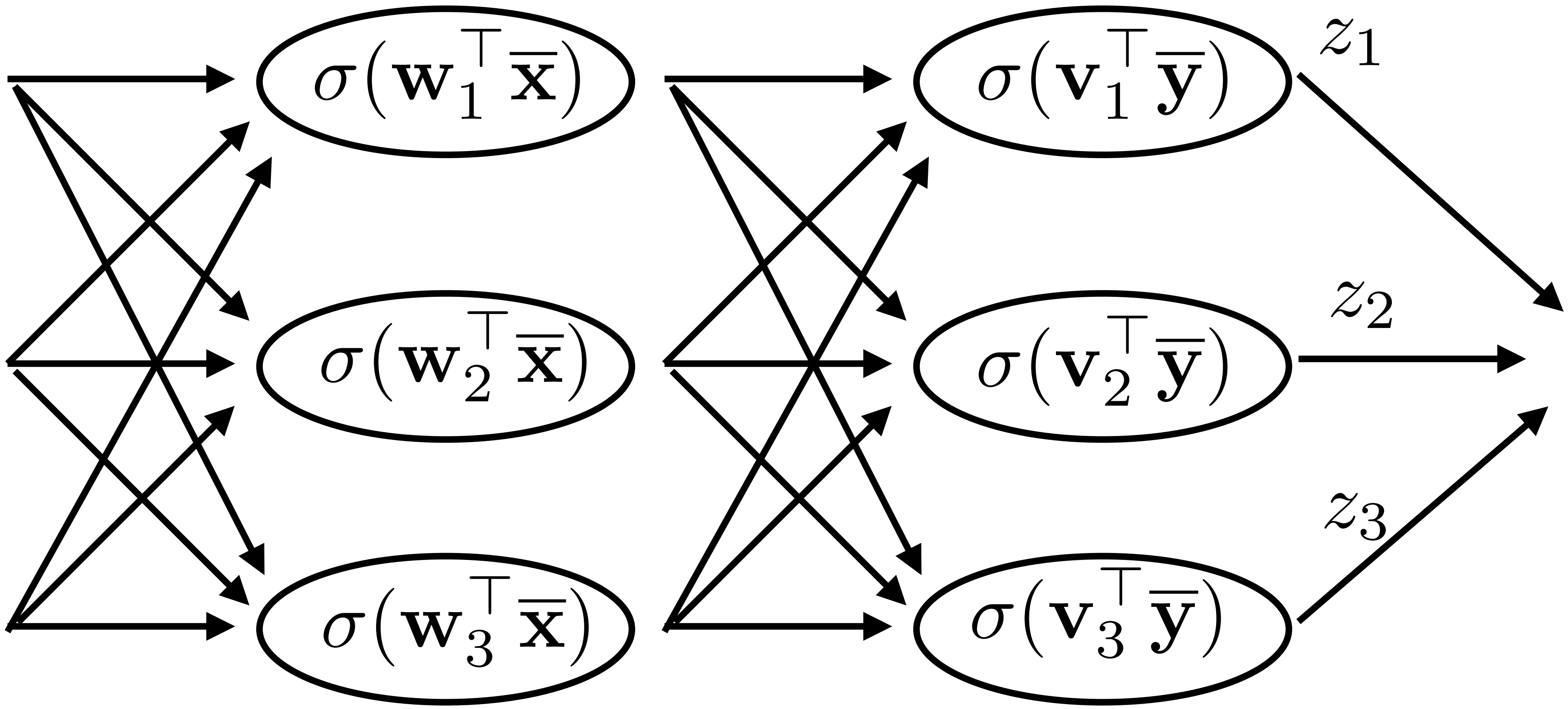
Multiple neurons and layers



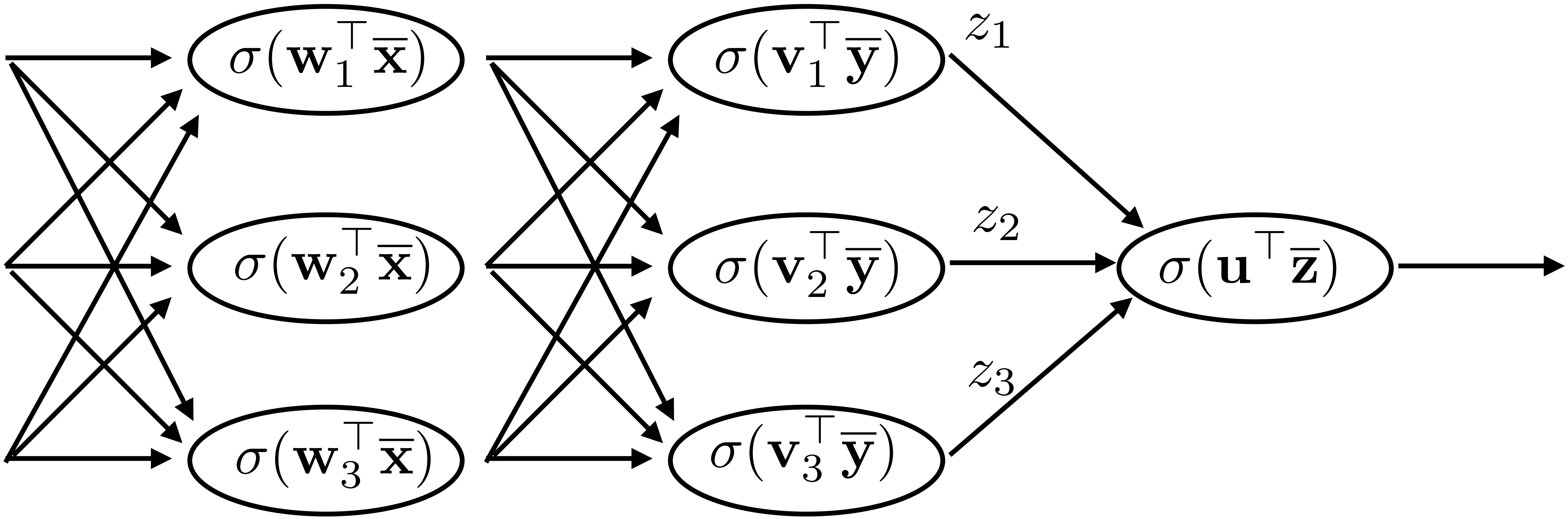
Multiple neurons and layers



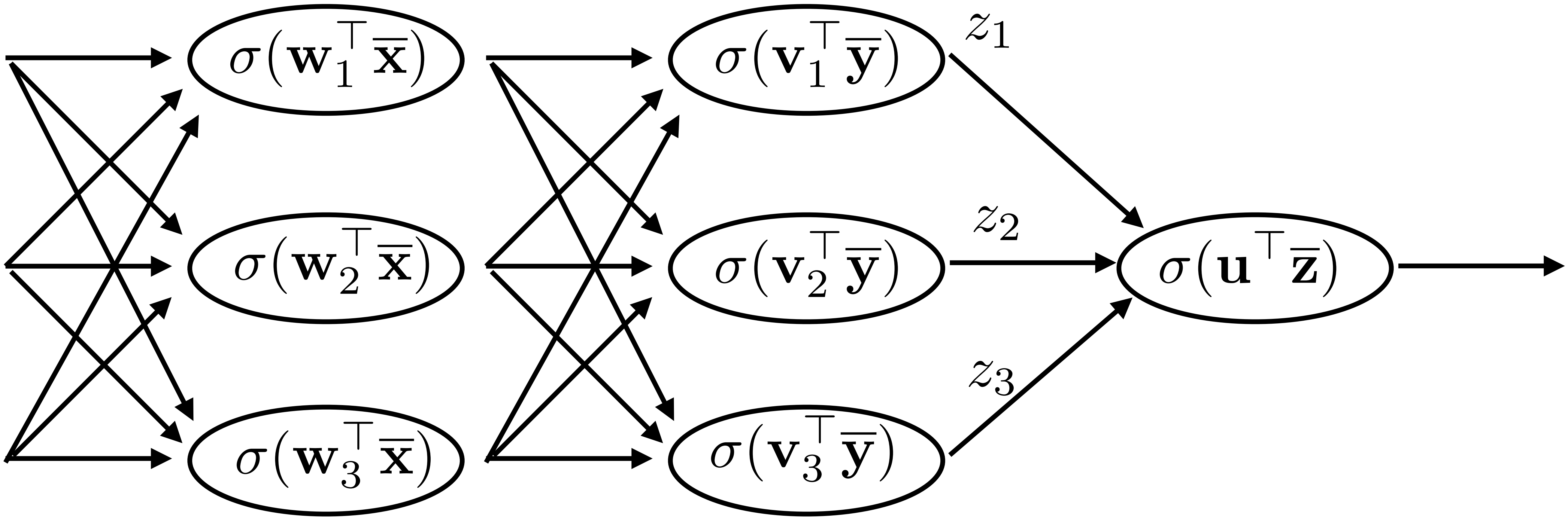
Multiple neurons and layers



Multiple neurons and layers

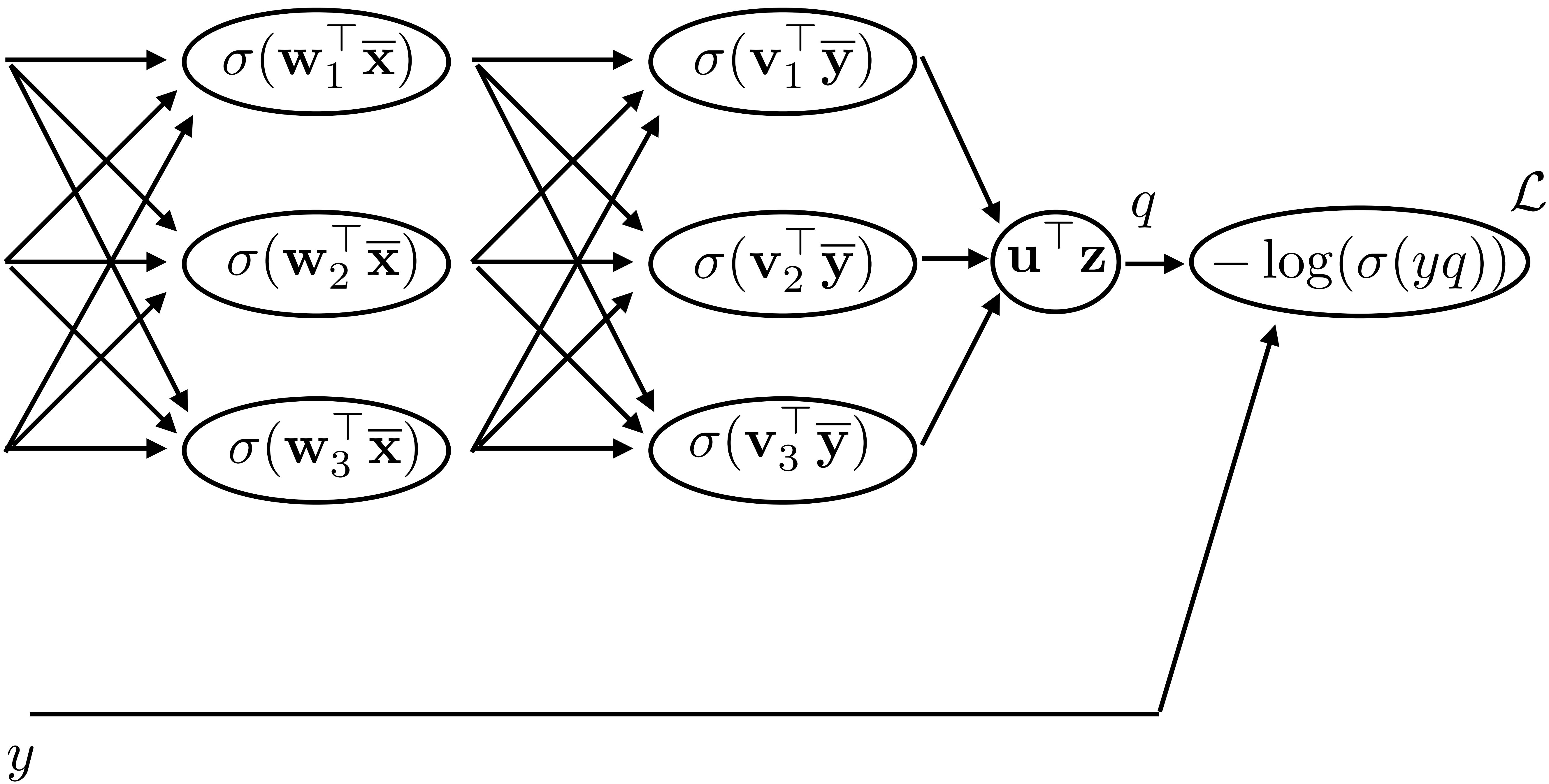


Multiple neurons and layers

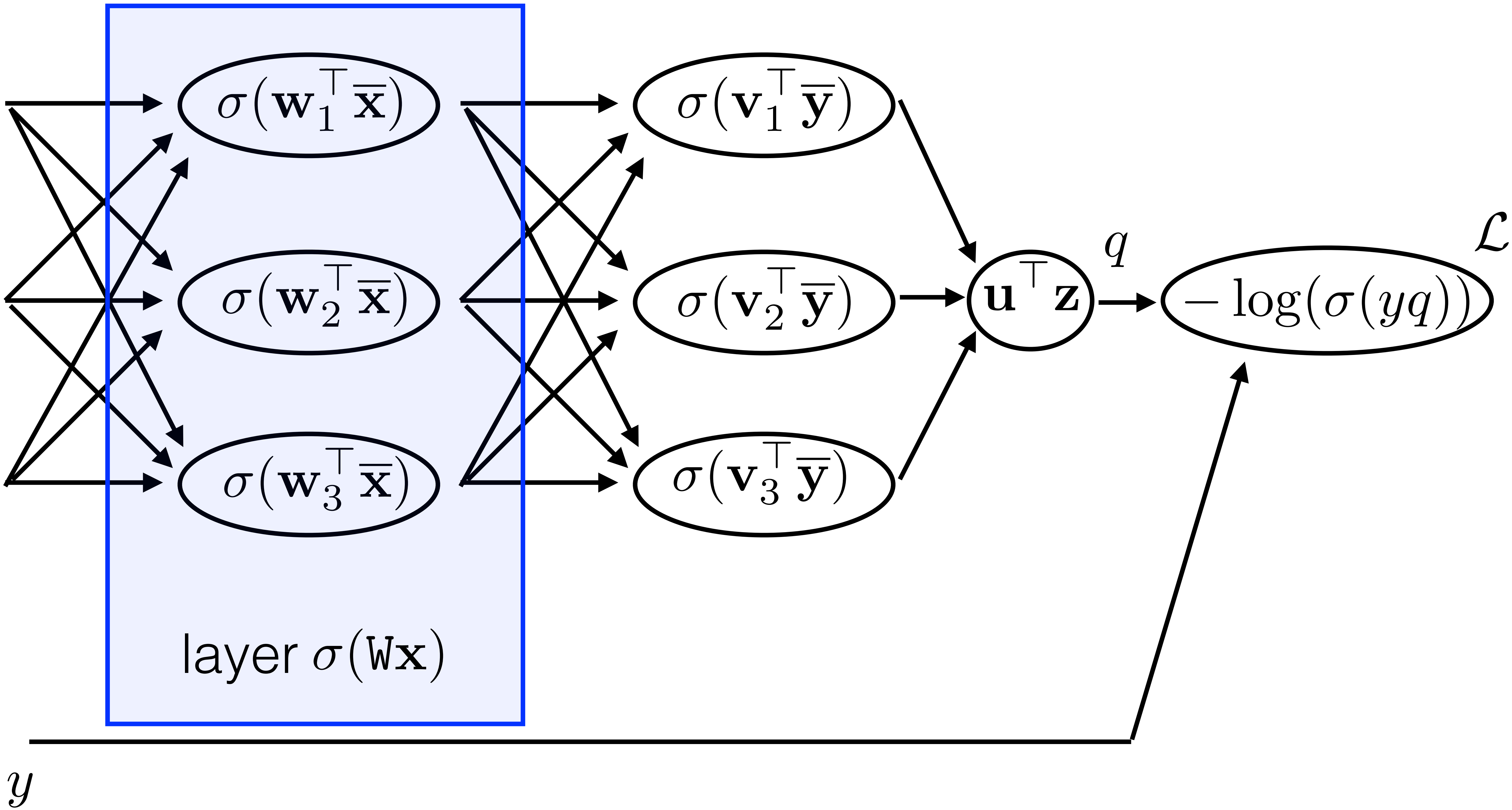


- What is dimensionality of weights?

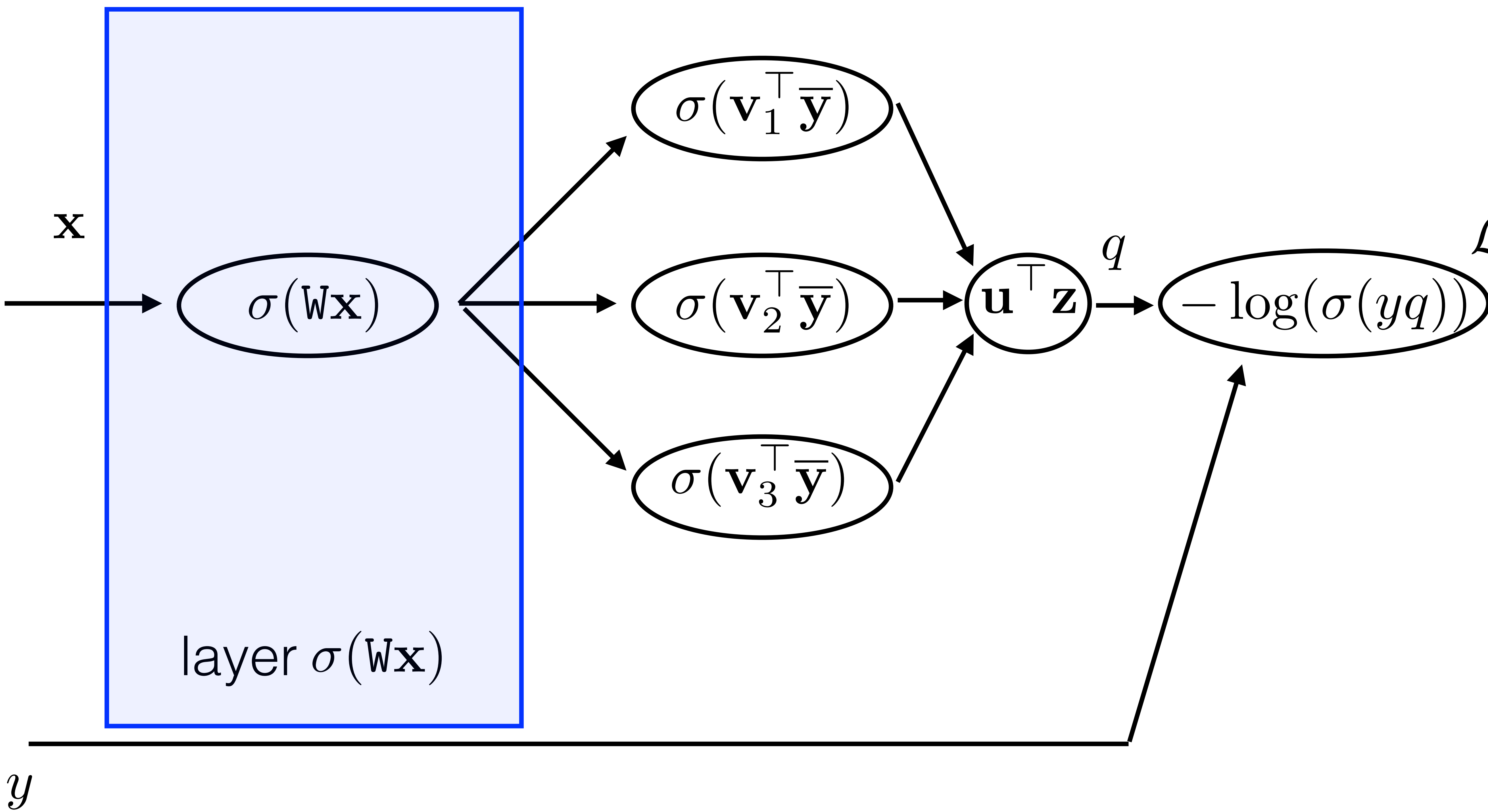
Learning of fully connected neural network



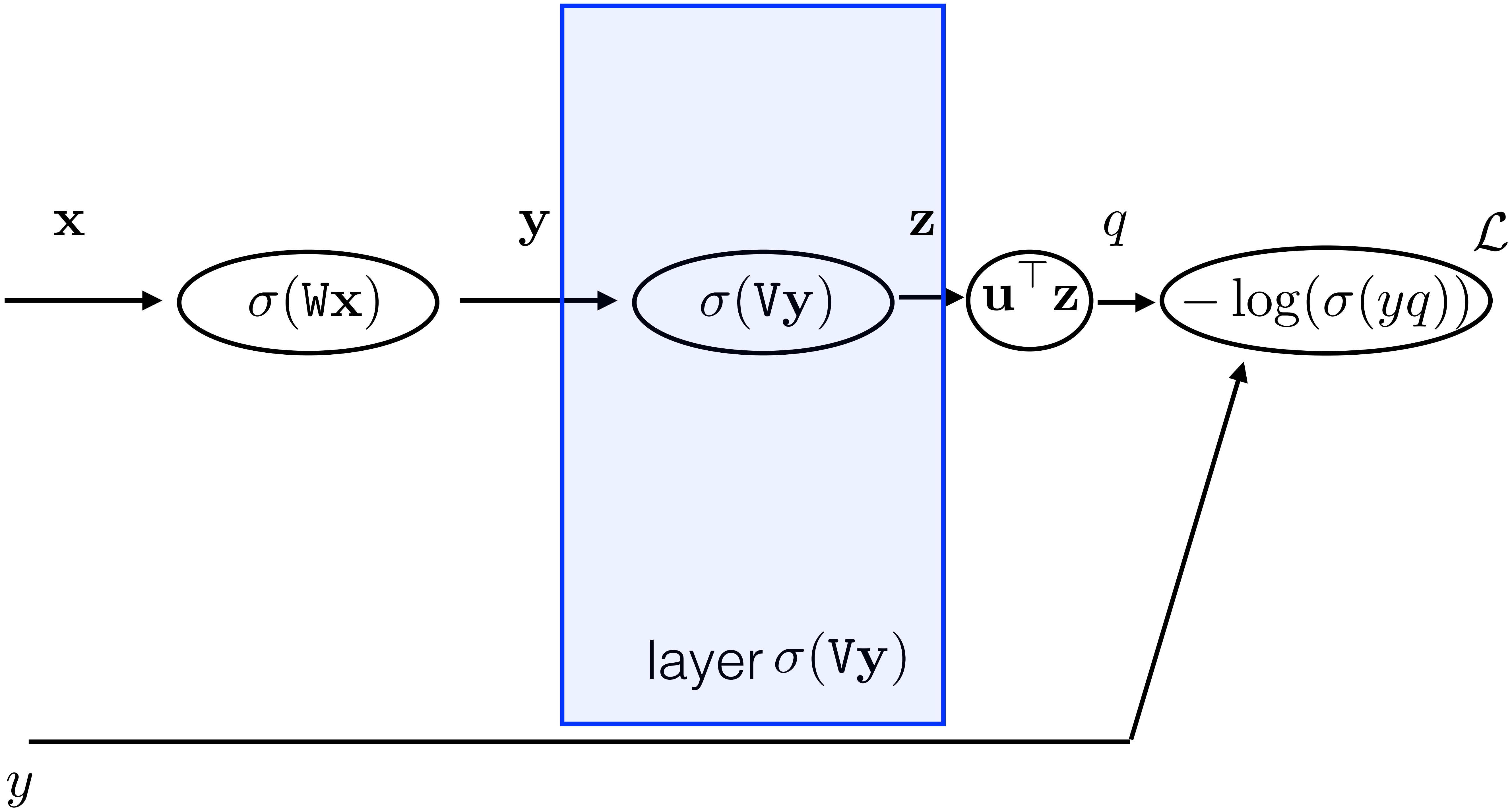
Learning of fully connected neural network



Learning of fully connected neural network

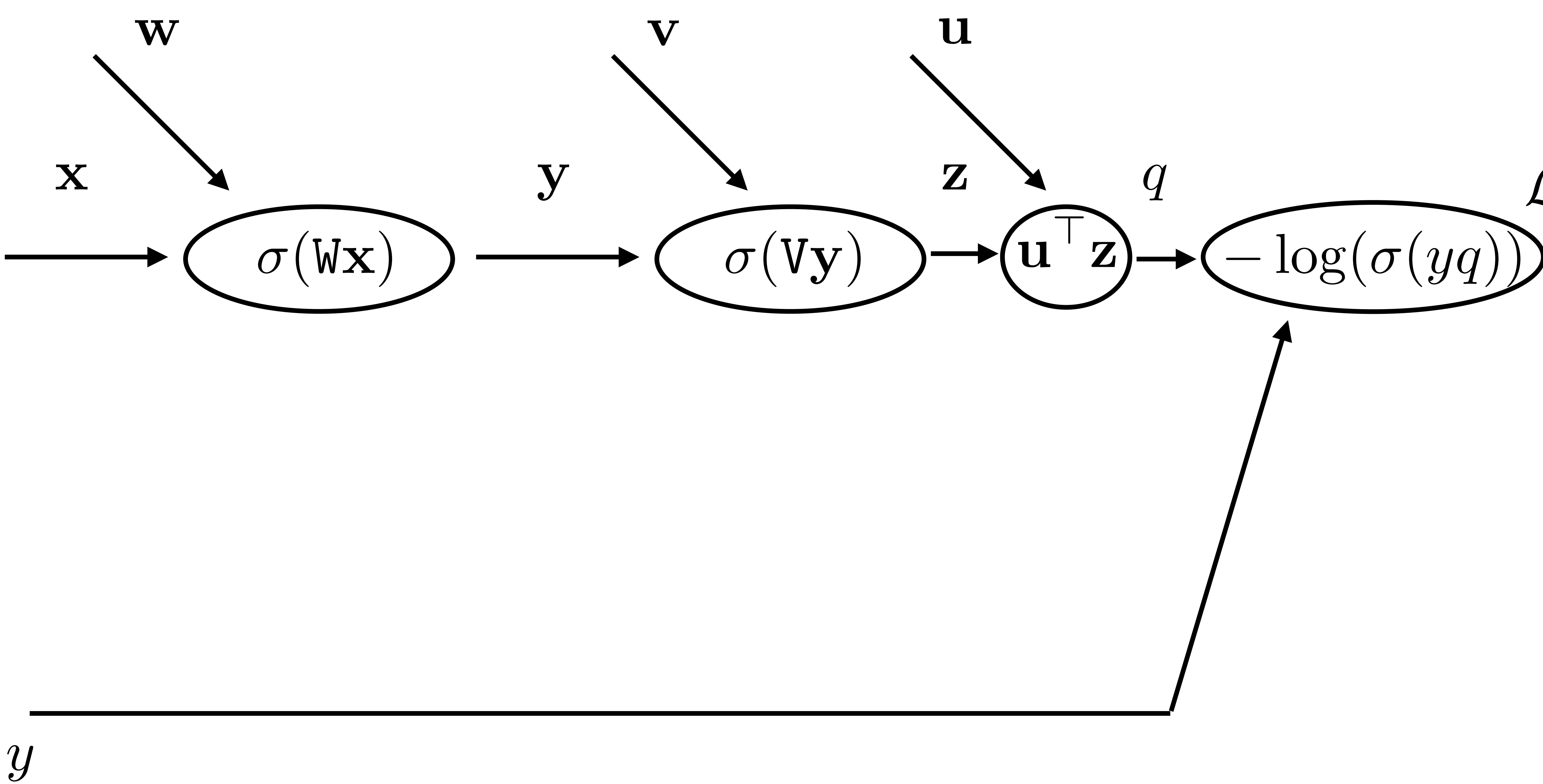


Learning of fully connected neural network



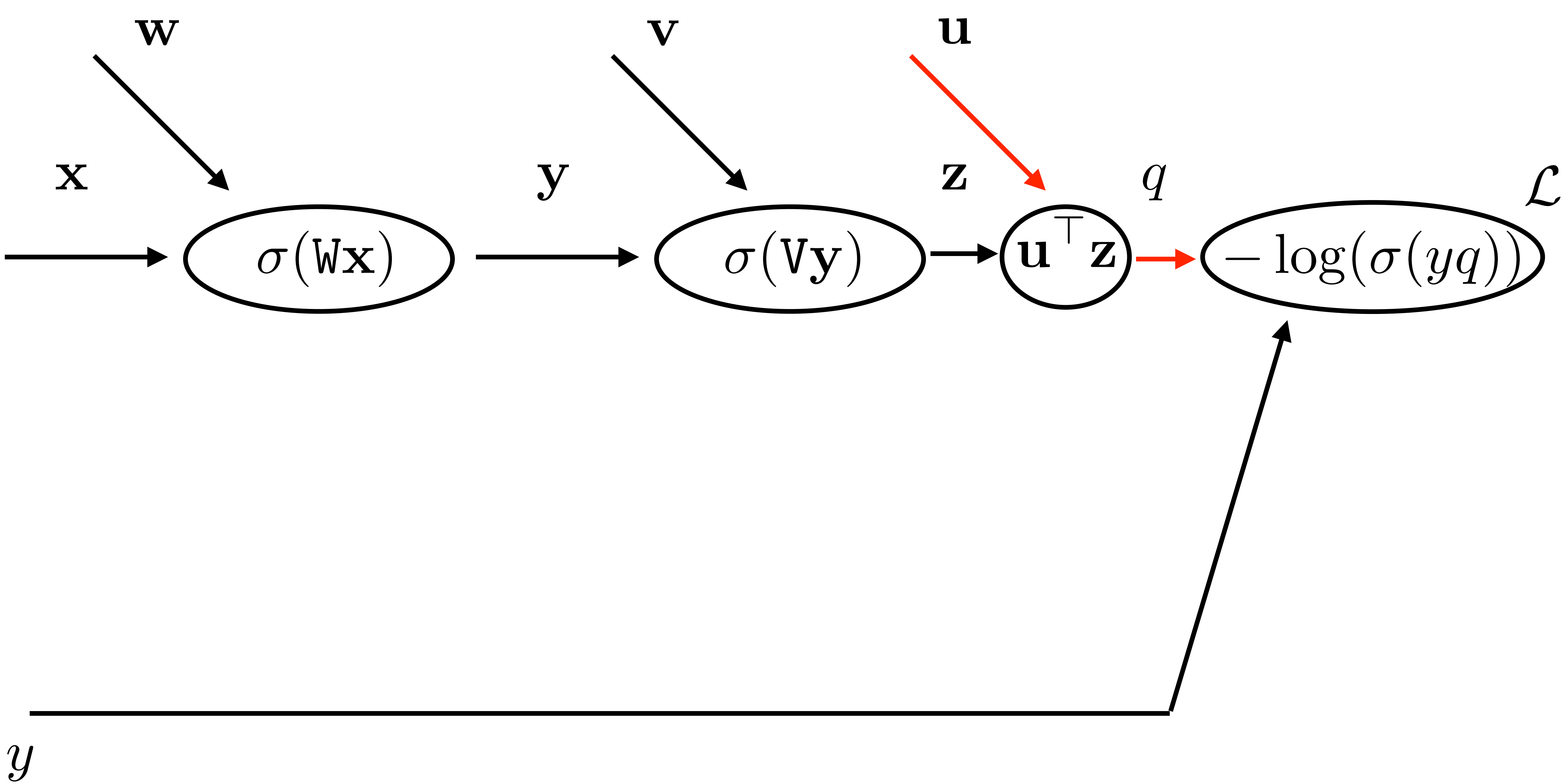
Learning of fully connected neural network

$\mathbf{w} = \text{vec}(\mathtt{W})$ $\qquad$ $\mathbf{v} = \text{vec}(\mathtt{V})$



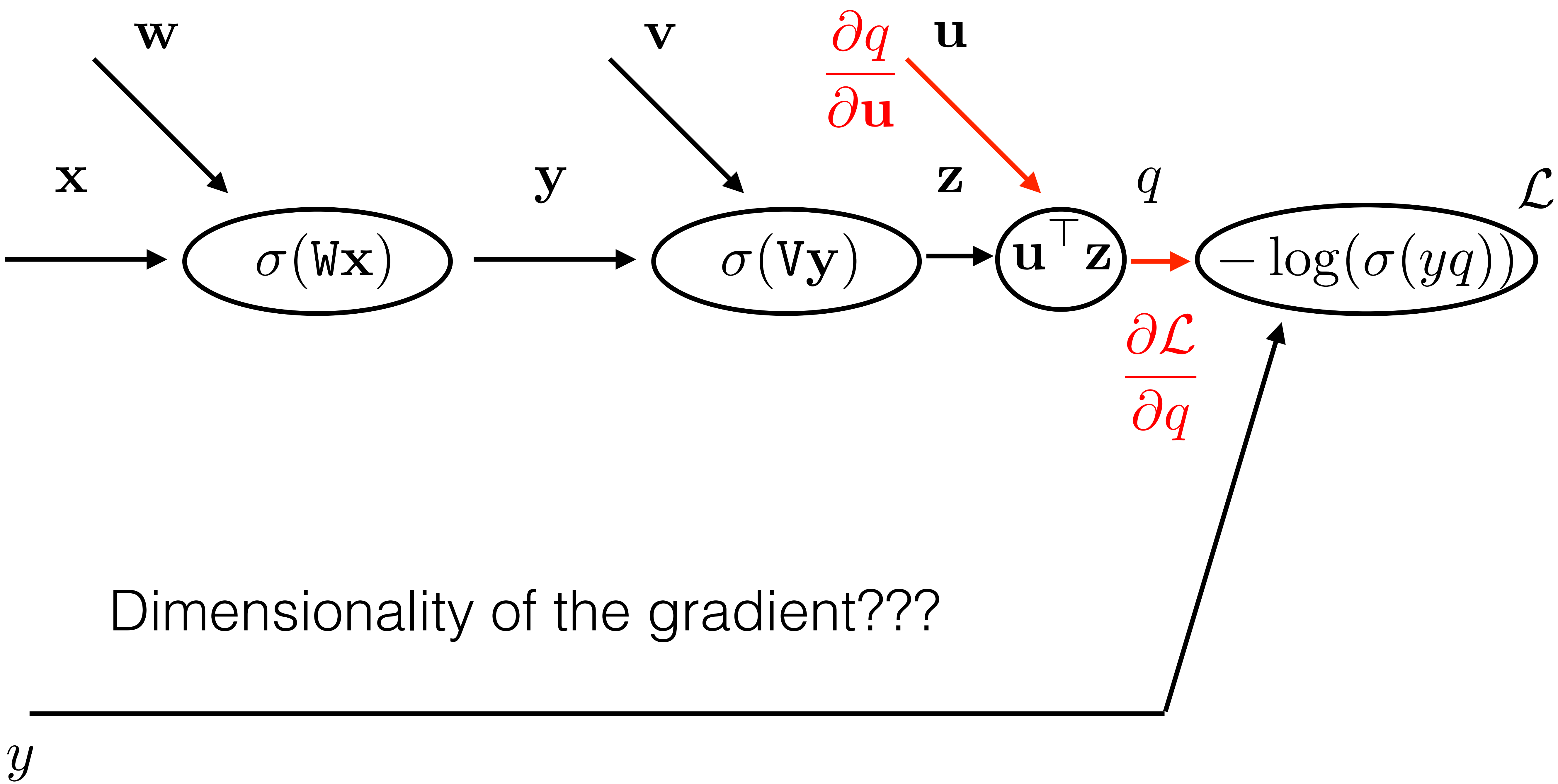
Learning of fully connected neural network

Derivative wrt $\mathbf{u}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = ?$



Learning of fully connected neural network

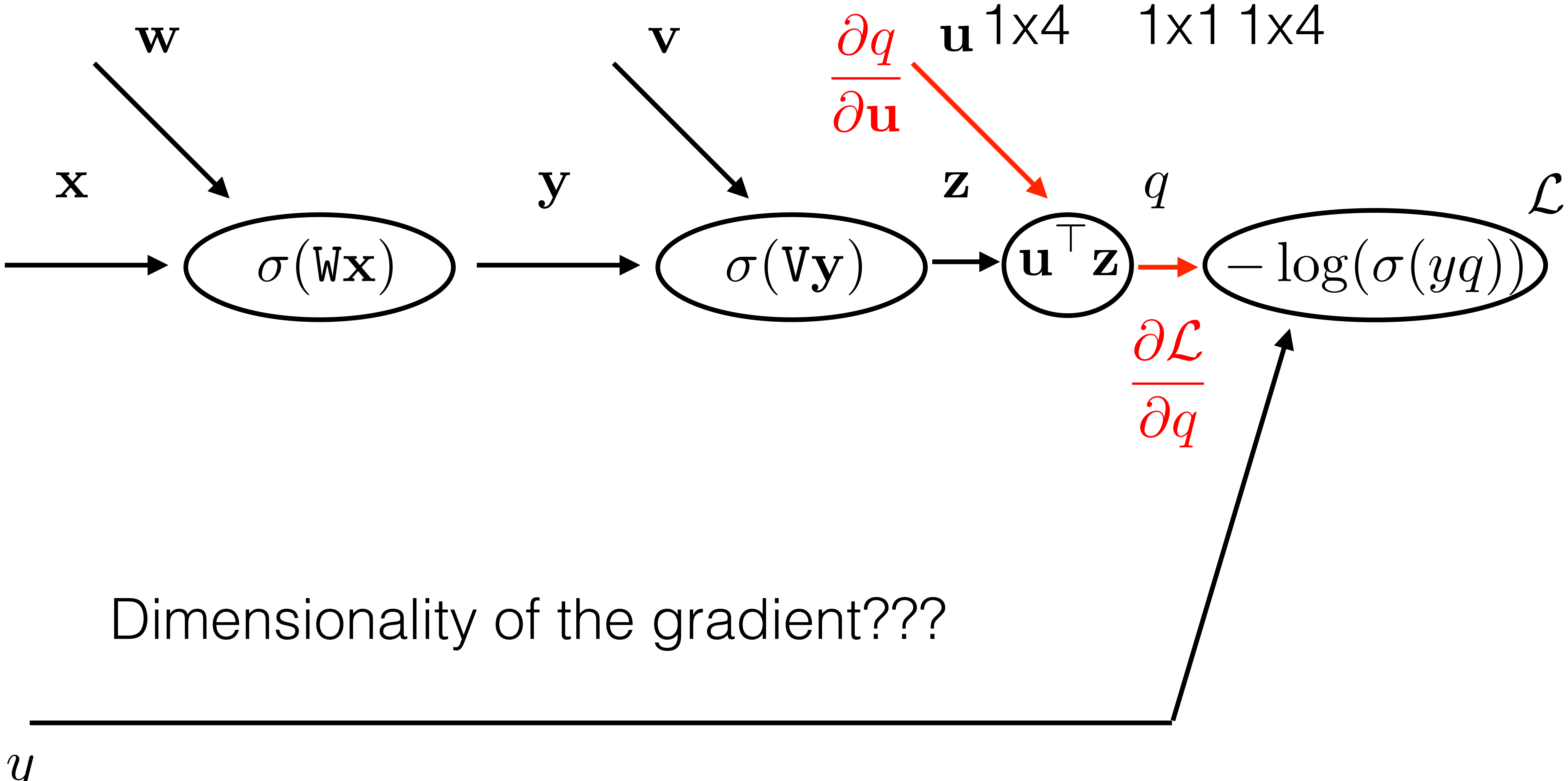
Derivative wrt $\mathbf{u}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{u}}$



Dimensionality of the gradient???

Learning of fully connected neural network

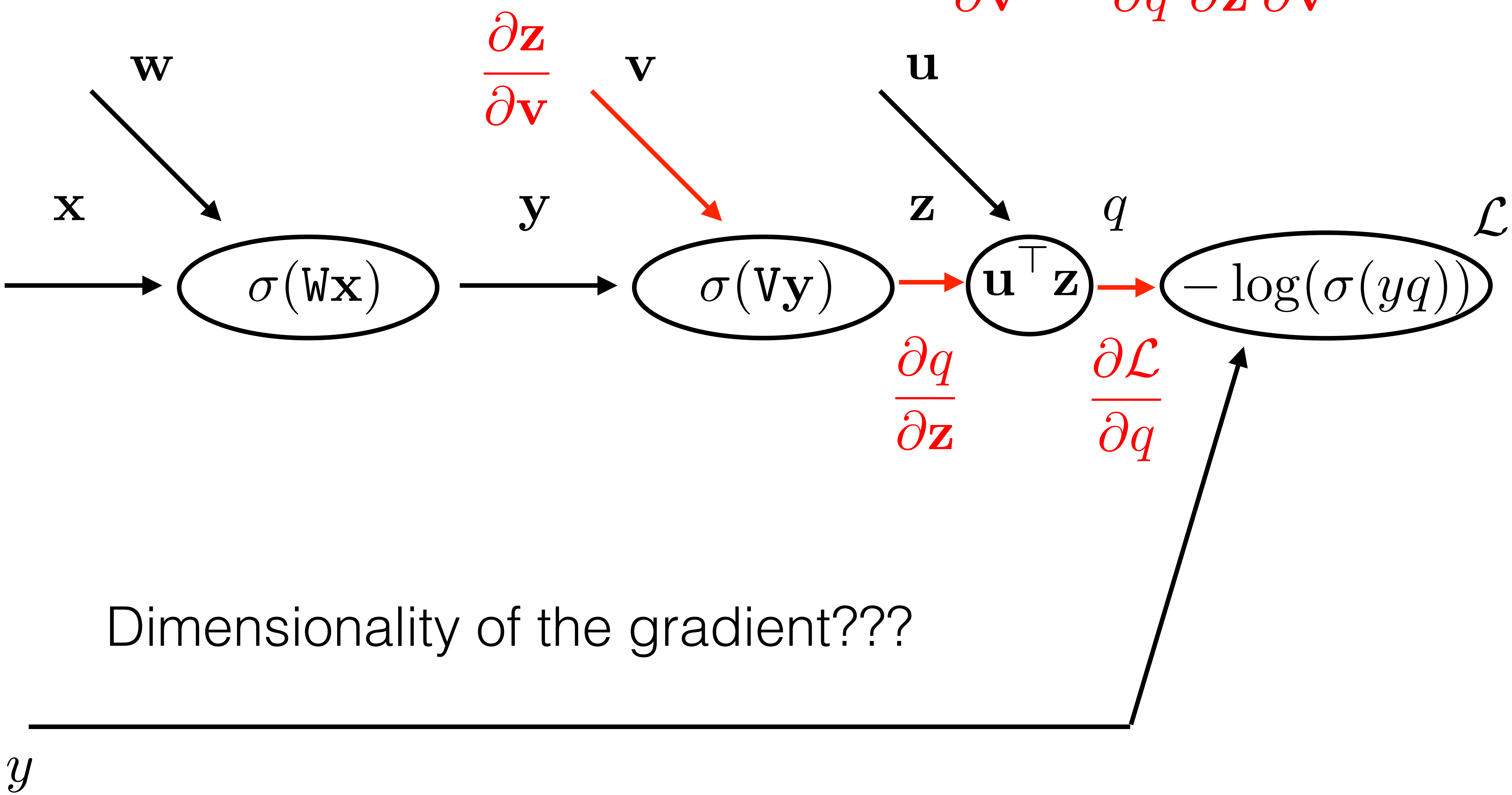
Derivative wrt $\mathbf{u}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{u}}$



Dimensionality of the gradient???

Learning of fully connected neural network

Derivative wrt $\mathbf{v}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{v}}$

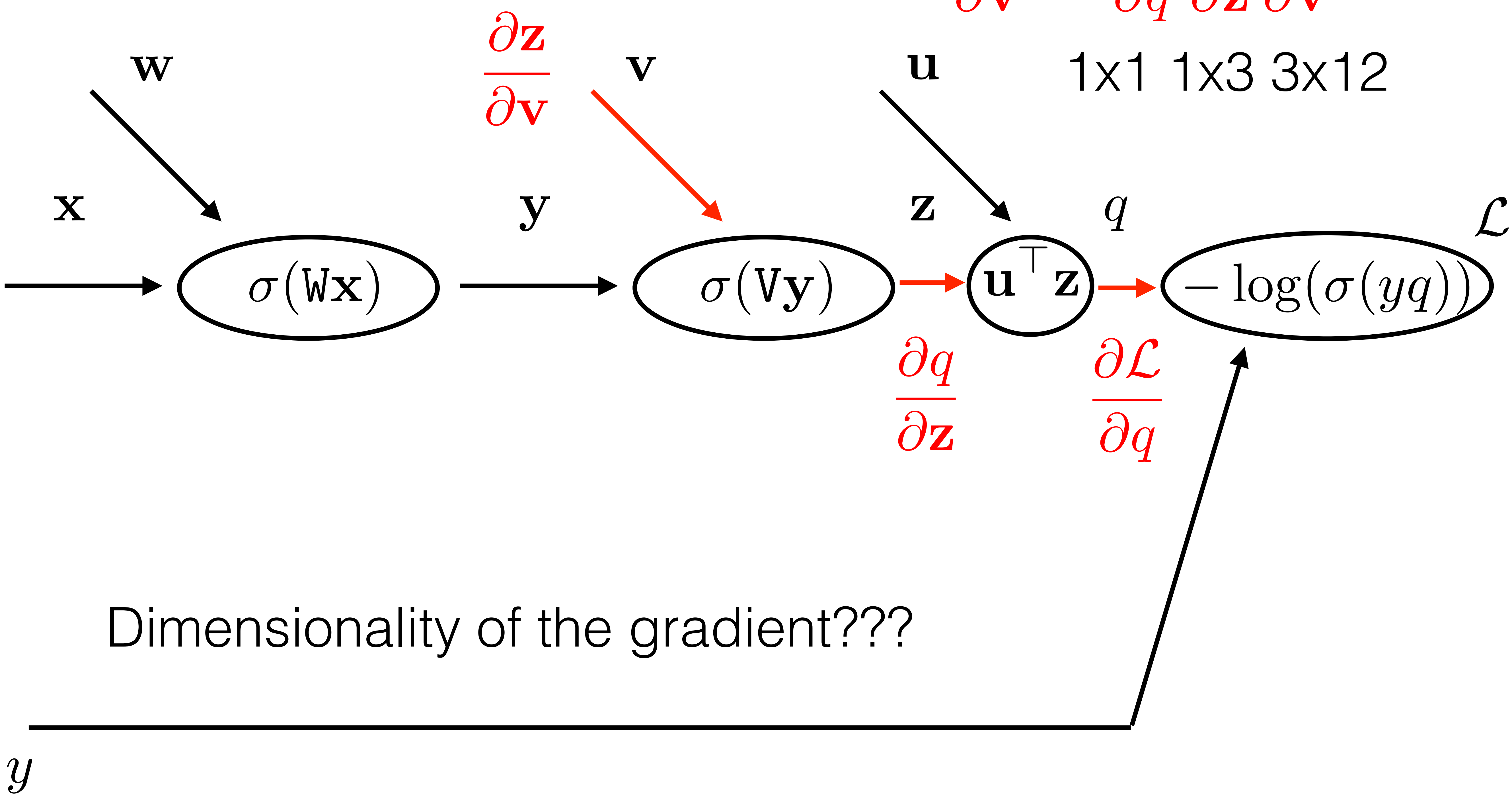


Dimensionality of the gradient???

Learning of fully connected neural network

Derivative wrt $\mathbf{v}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{v}}$

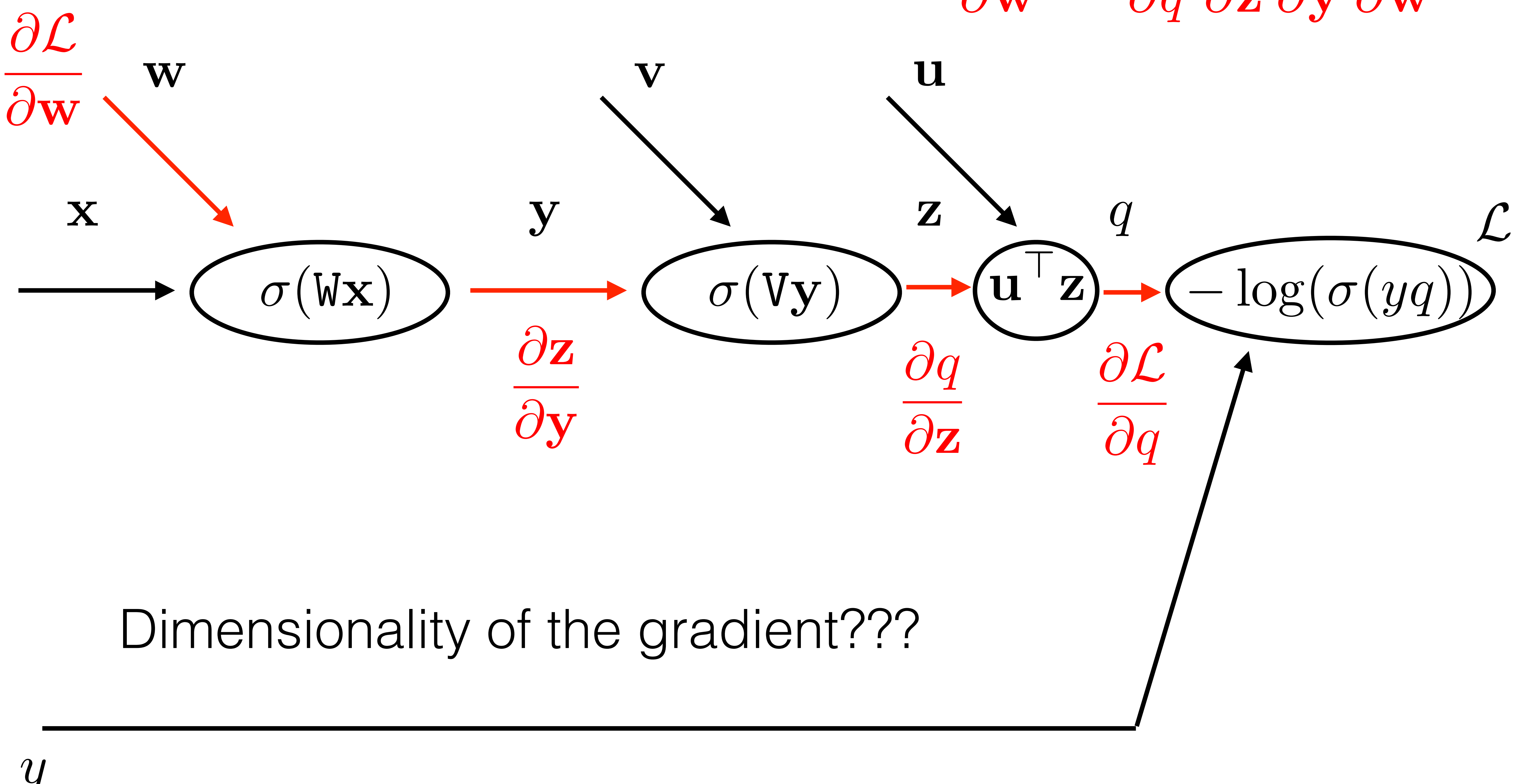
1x1 1x3 3x12



Dimensionality of the gradient???

Learning of fully connected neural network

Derivative wrt $\mathbf{w}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{w}}$

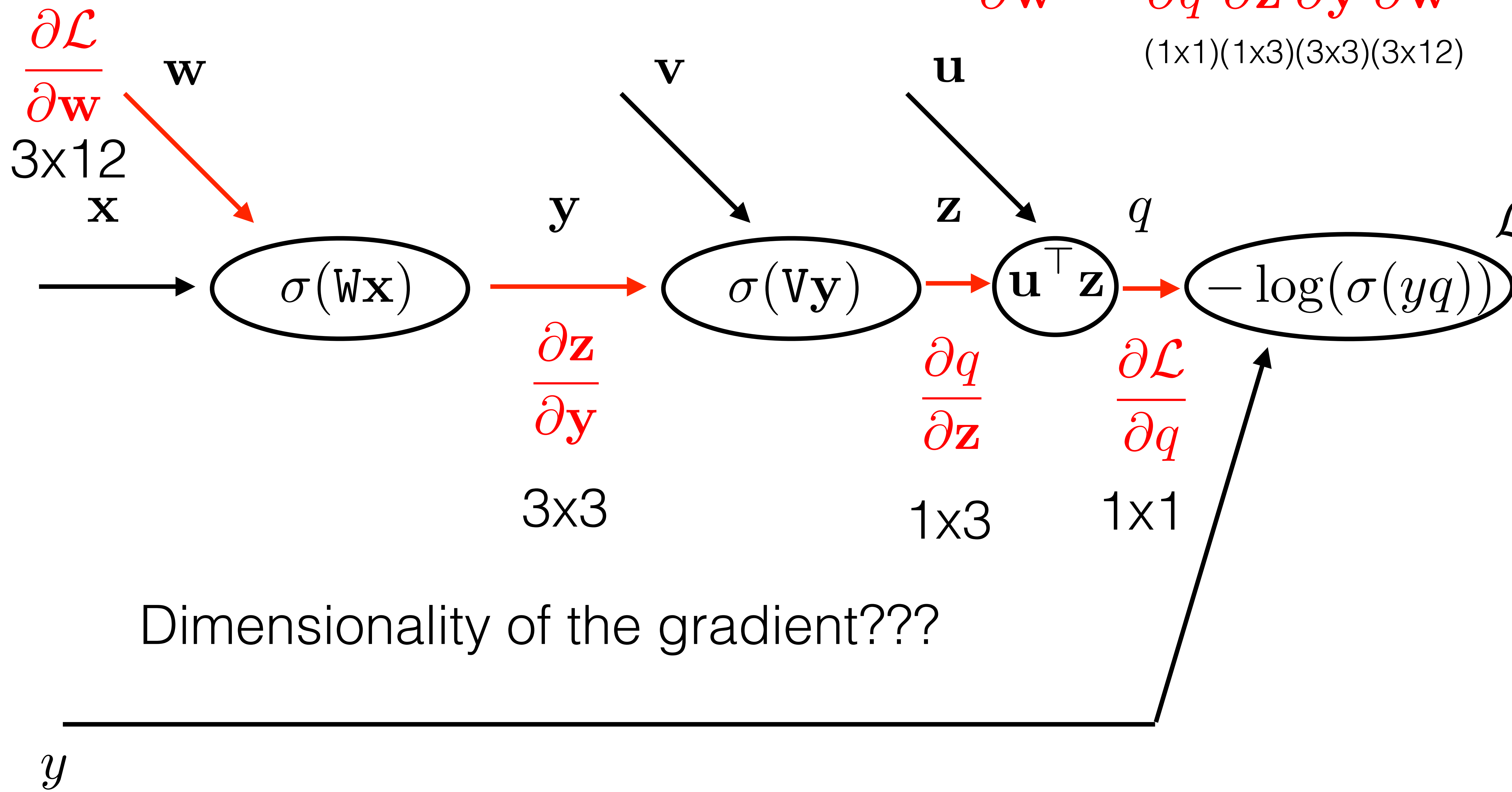


Dimensionality of the gradient???

Learning of fully connected neural network

Derivative wrt $\mathbf{w}$: $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial q} \frac{\partial q}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{w}}$

(1x1)(1x3)(3x3)(3x12)



Dimensionality of the gradient???

Learning of fully connected neural network

1. Estimate all required local gradients
2. Update weights:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = \frac{\partial \mathcal{L}}{\partial q}\frac{\partial q}{\partial \mathbf{u}} \qquad \mathbf{u} = \mathbf{u} - \alpha \left[\frac{\partial \mathcal{L}}{\partial \mathbf{u}}\right]^\top$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}} = \frac{\partial \mathcal{L}}{\partial q}\frac{\partial q}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{v}} \qquad \mathbf{v} = \mathbf{v} - \alpha \left[\frac{\partial \mathcal{L}}{\partial \mathbf{v}}\right]^\top$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial q}\frac{\partial q}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{w}} \qquad \mathbf{w} = \mathbf{w} - \alpha \left[\frac{\partial \mathcal{L}}{\partial \mathbf{w}}\right]^\top$$

3. Optionally update learning rate α
4. Repeat until convergence

Neural nets summary

- Neural net is a function created as concatenation of simplier functions (e.g. neurons or layers of neurons)
- Gradient optimization of the neural net is called backpropagation
- Neural net frameworks has many predefined layers
- **Spoiler alert:** It does not work (on images) at all - why?

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left(\sum_i -\log(p(y_i|\mathbf{x}_i, \mathbf{w})) \right) + (-\log p(\mathbf{w}))$$

loss function prior/regulariser

- Class of function represented by a NN is too general.
- Naive regulariser helps a bit, but dimensionality/wildness is huge => curse-of-dimensionality, overfitting,…
- What is number of weights between two 1000-neuron layers?
- **Next lecture:** study animal cortex to find a stronger prior on the class of suitable functions.
- **Spoiler alert 2:**
  reduce very general class of functions ”neuron layer” to very specific sub-class of functions “convolution layer”