Lecture 8:
Multilayer Perceptrons

Artificial Intelligence
CS-GY-6613

~~Julian Togelius~~
~~julian.togelius@nyu.edu~~

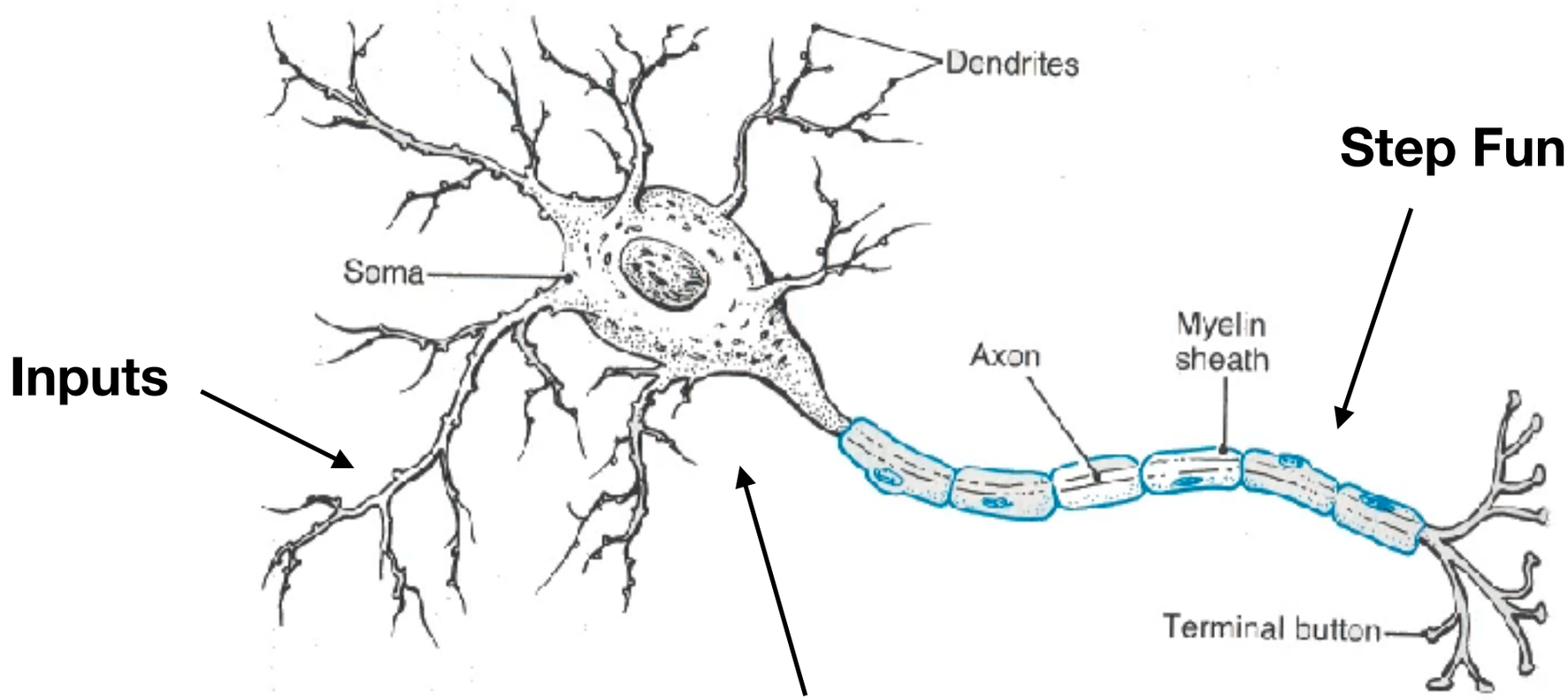
Philip Bontrager
2019

Contents

- Review Perceptrons
- Multilayer Perceptron Intro
- Computational Graphs
- How to Train a Multilayer Perceptron
- Why Deep Learning now

Perceptron

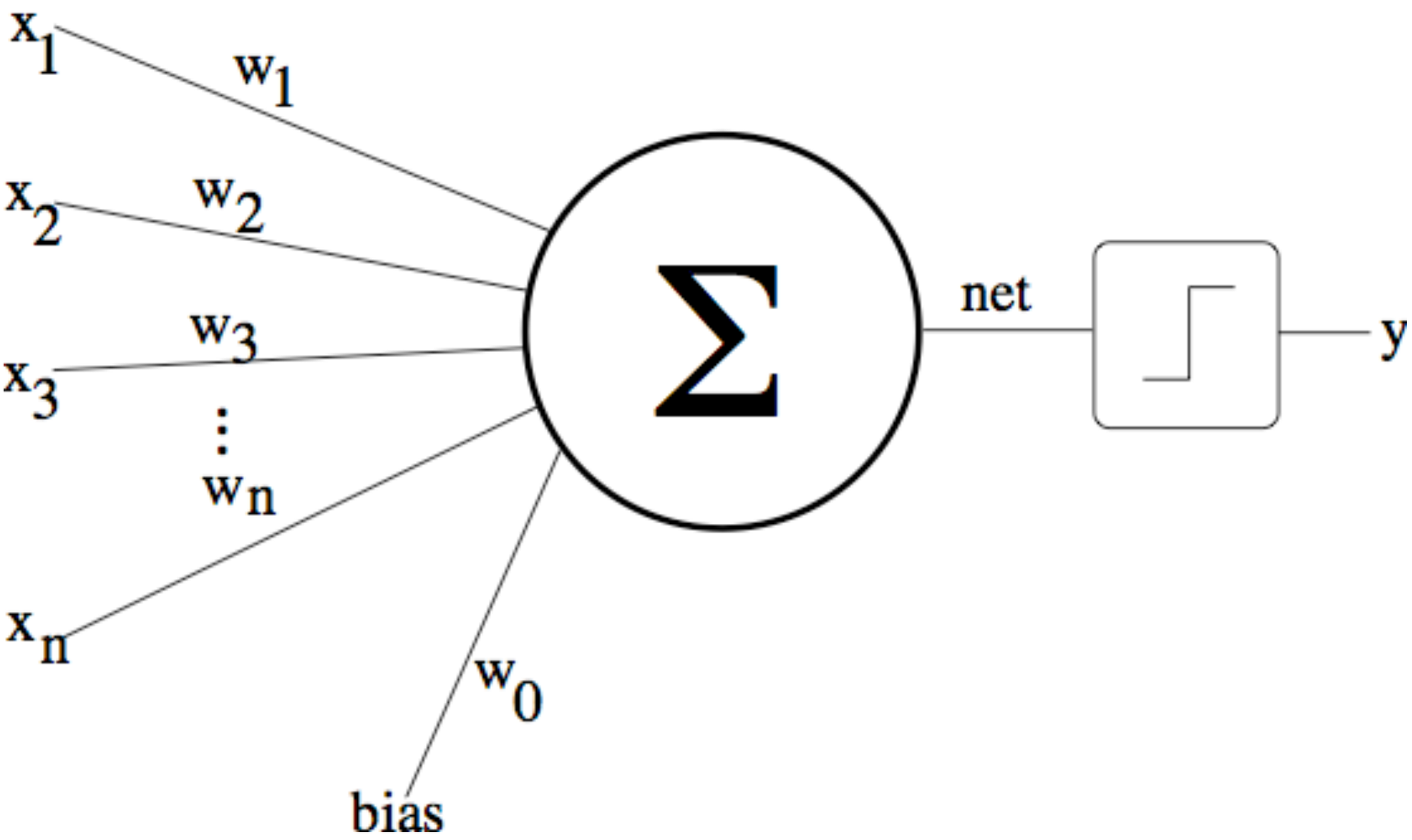
The Neuron



Inputs

Step Function

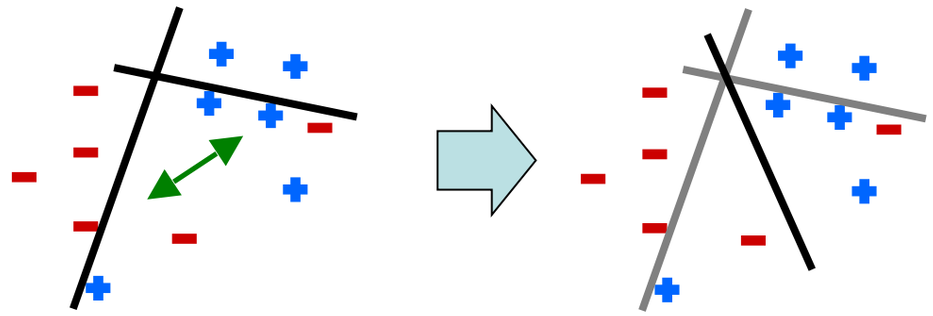
Sum Together



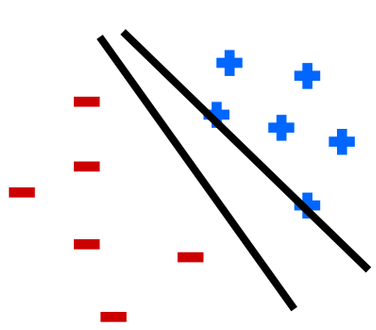
$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0 & \text{else} \end{cases}$$

Perceptron problems

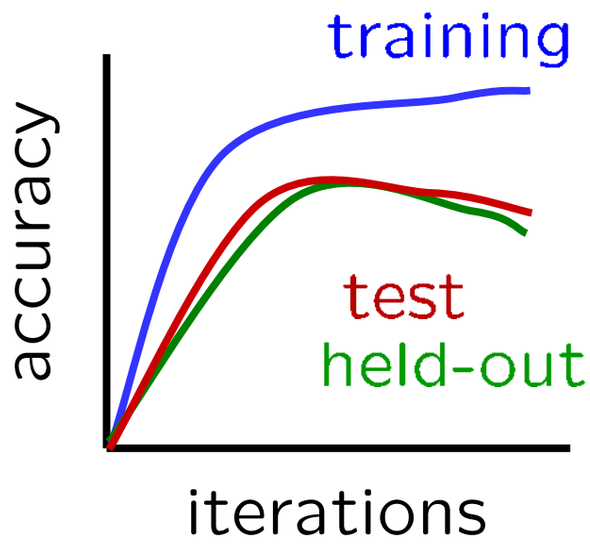
- Noise: if the data isn't separable, weights might thrash



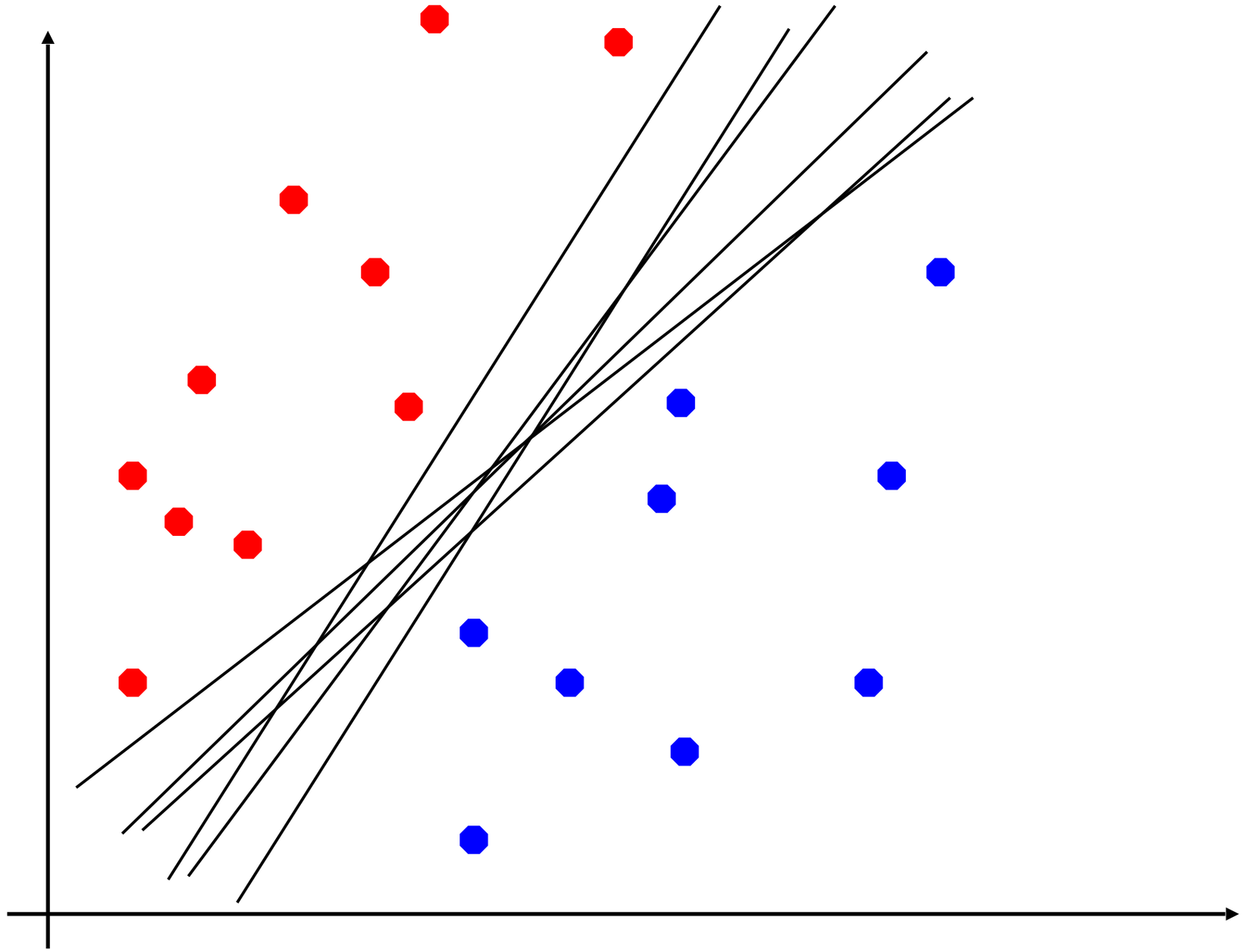
- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls



Which of these separators is optimal?



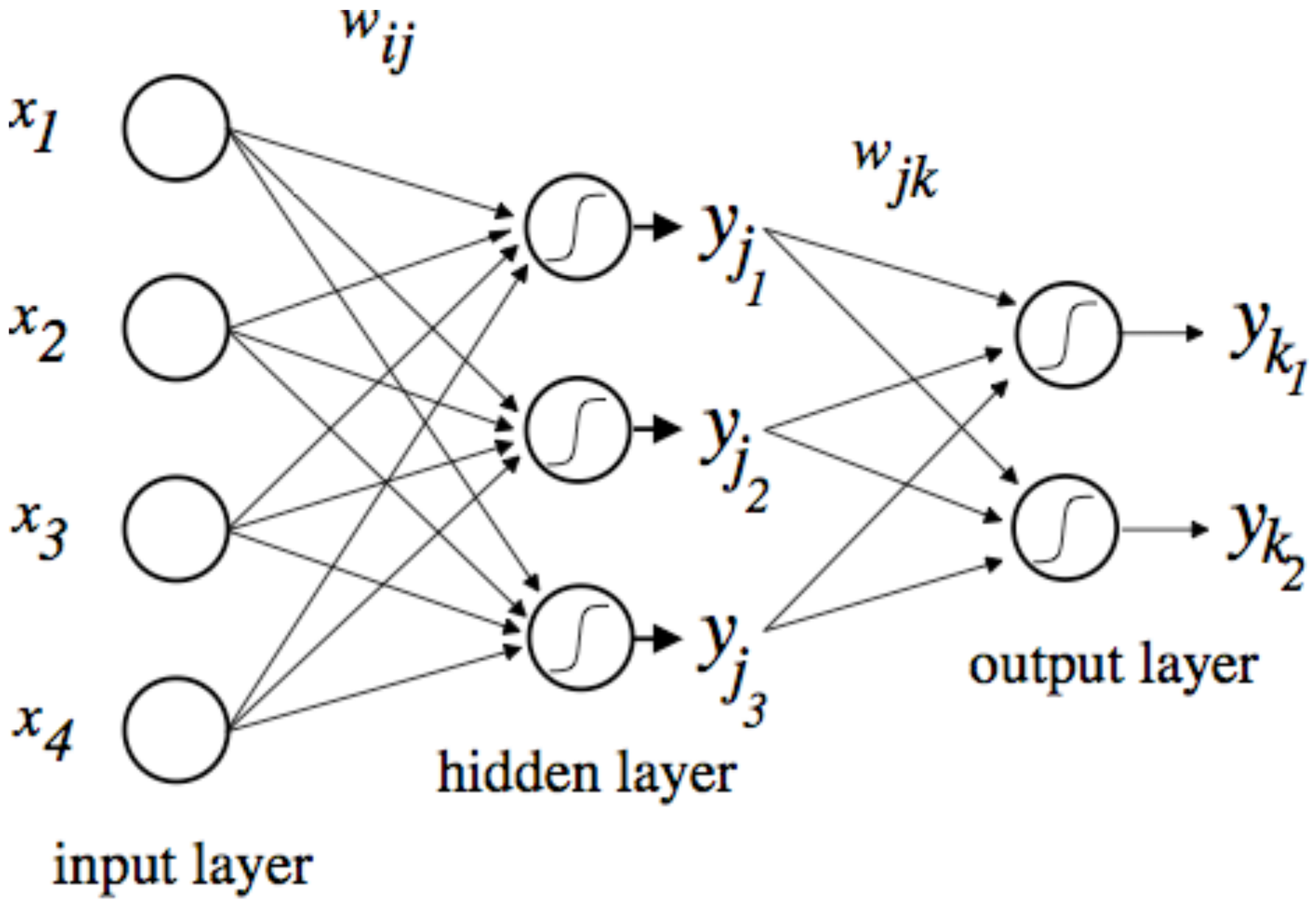
With all of these problems and with so many other learning algorithms, why were people still interested in pursuing this technique?

Multilayer Perceptron

“When an axon of cell A is near enough cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased”

–Donald Hebb

Multi-layer Perceptron (MLP)



Universal Approximator

- George Cybenko in 1989 provided one of the first proofs for the Universal Approximation Theorem for Neural Networks

- States that a feedforward network with at least one hidden layer of a finite size can approximate any continuous function on compact subsets of R^n

- Says nothing about learnability

    - How do you train a hidden layer? What is its loss?

Computational Graphs

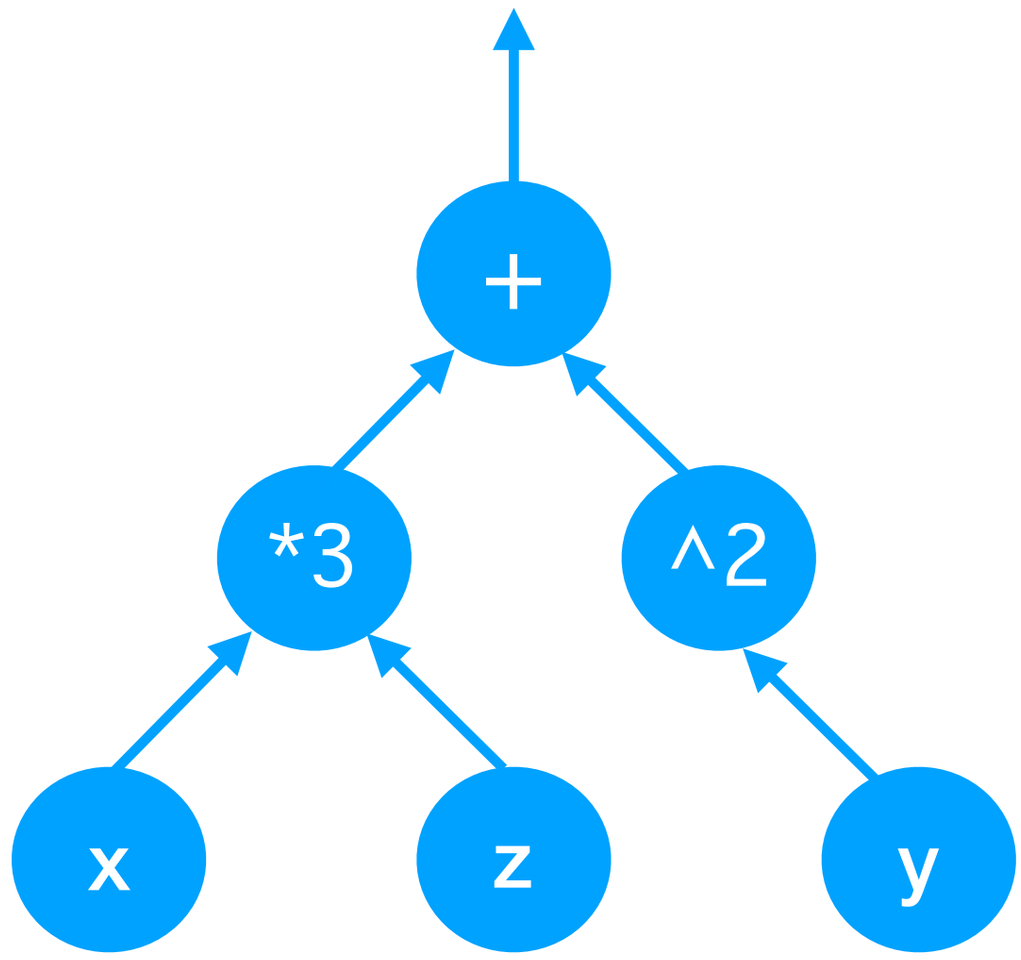
Section inspired by slides from Fei-Fei Li & Justin Johnson & Serena Yeung

Quick Calculus Refresh

- Partial Derivatives
  - $f(x, y, z) = 3xz + y^2$
  - $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} = 3z, 2y, 3x$
- Chain Rule
  - $(f \circ g)' = (f' \circ g) \cdot g'$

Functions as Computation Graphs

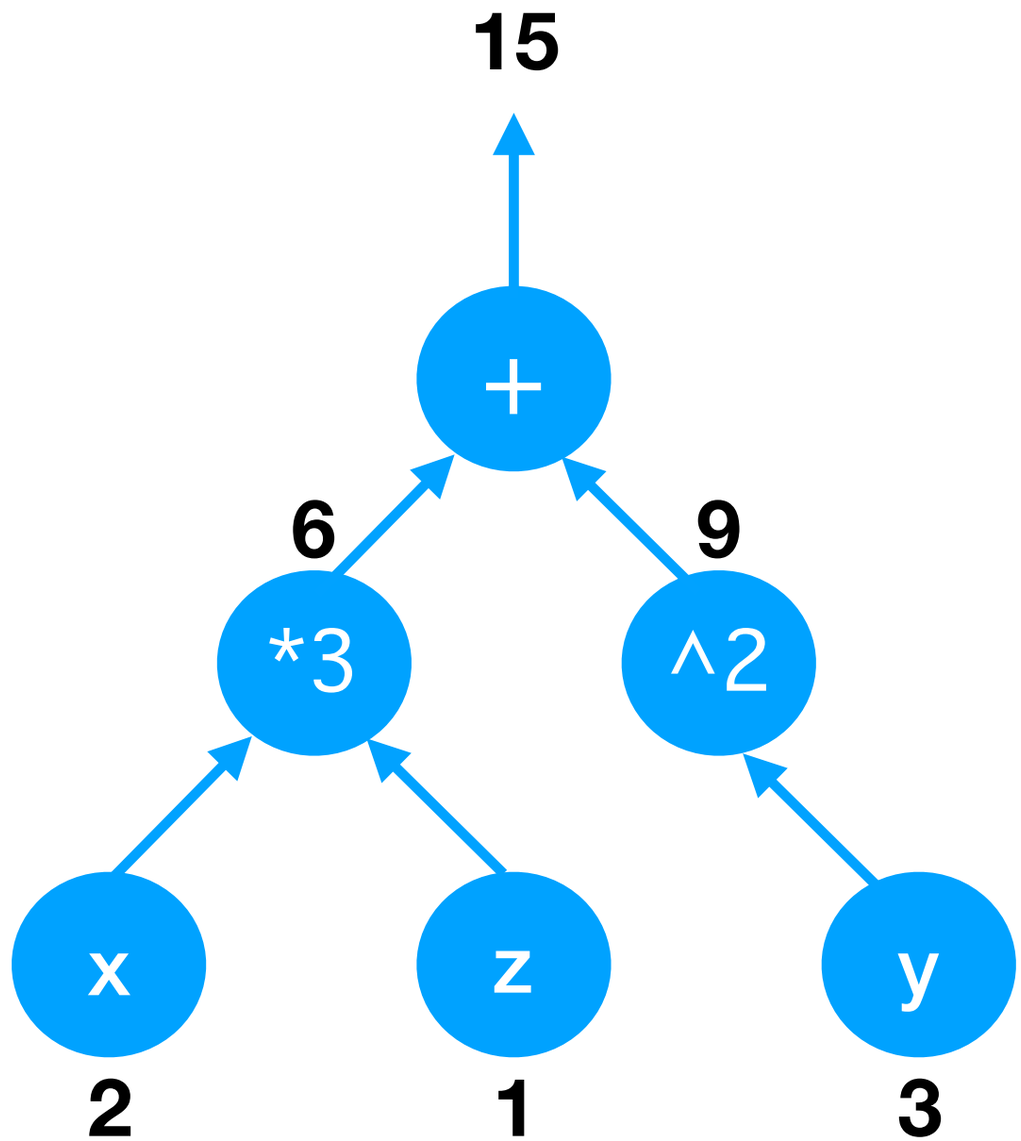
$$f(x, y, z) = 3xz + y^2$$



Functions as Computation Graphs

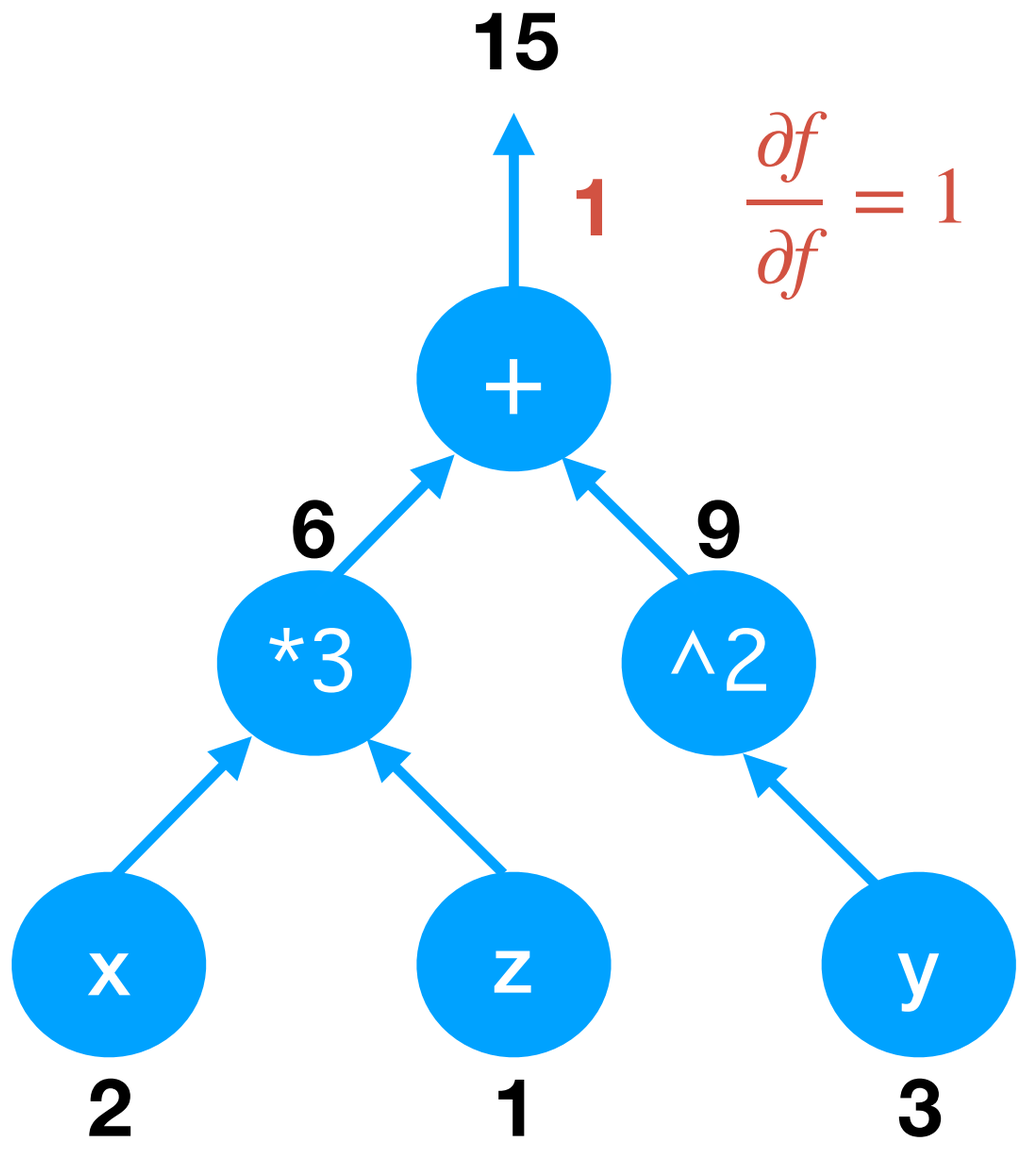
$$f(x, y, z) = 3xz + y^2$$

$$x = 2,\ y = 3,\ z = 1$$

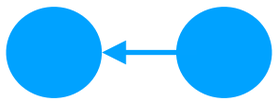


Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$
$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:



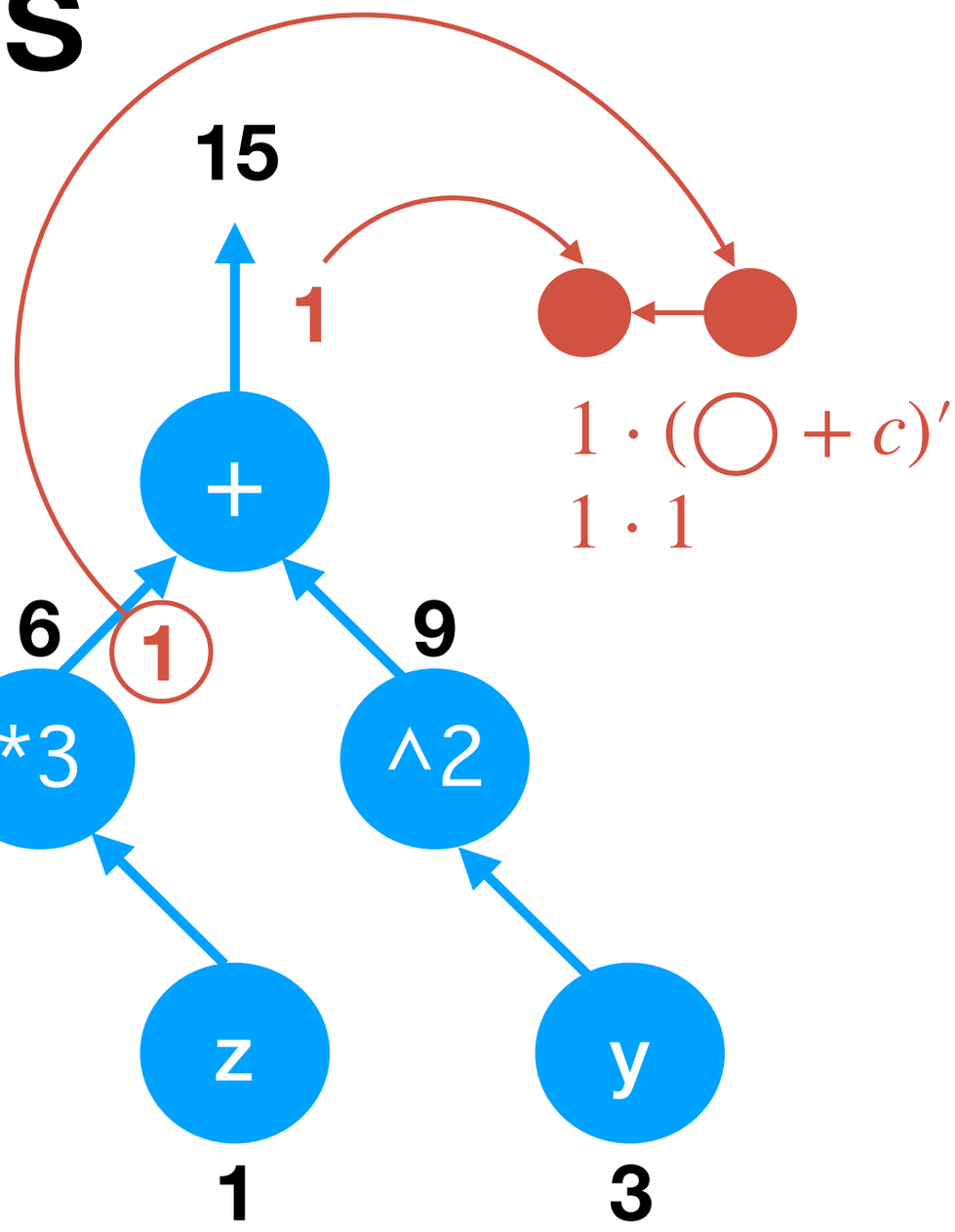
$$(f \circ g)' = (f' \circ g) \cdot g'$$

Encapsulating Function Gradient

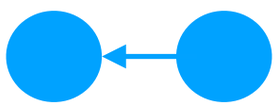
Node Gradient

Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$
$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:



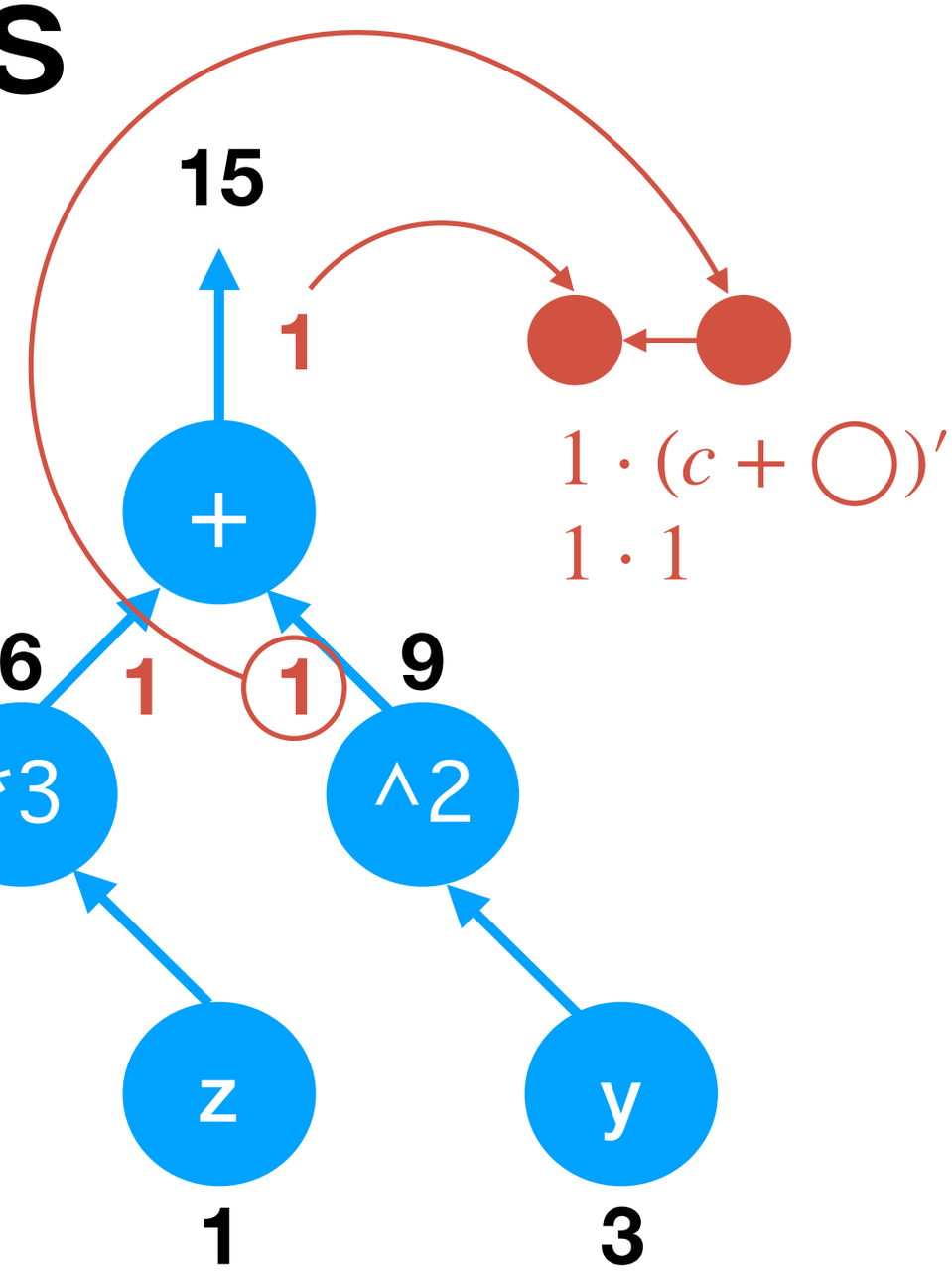
$$(f \circ g)' = (f' \circ g) \cdot g'$$

Encapsulating Function Gradient

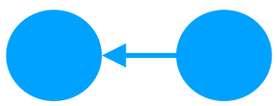
Node Gradient

Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$
$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:



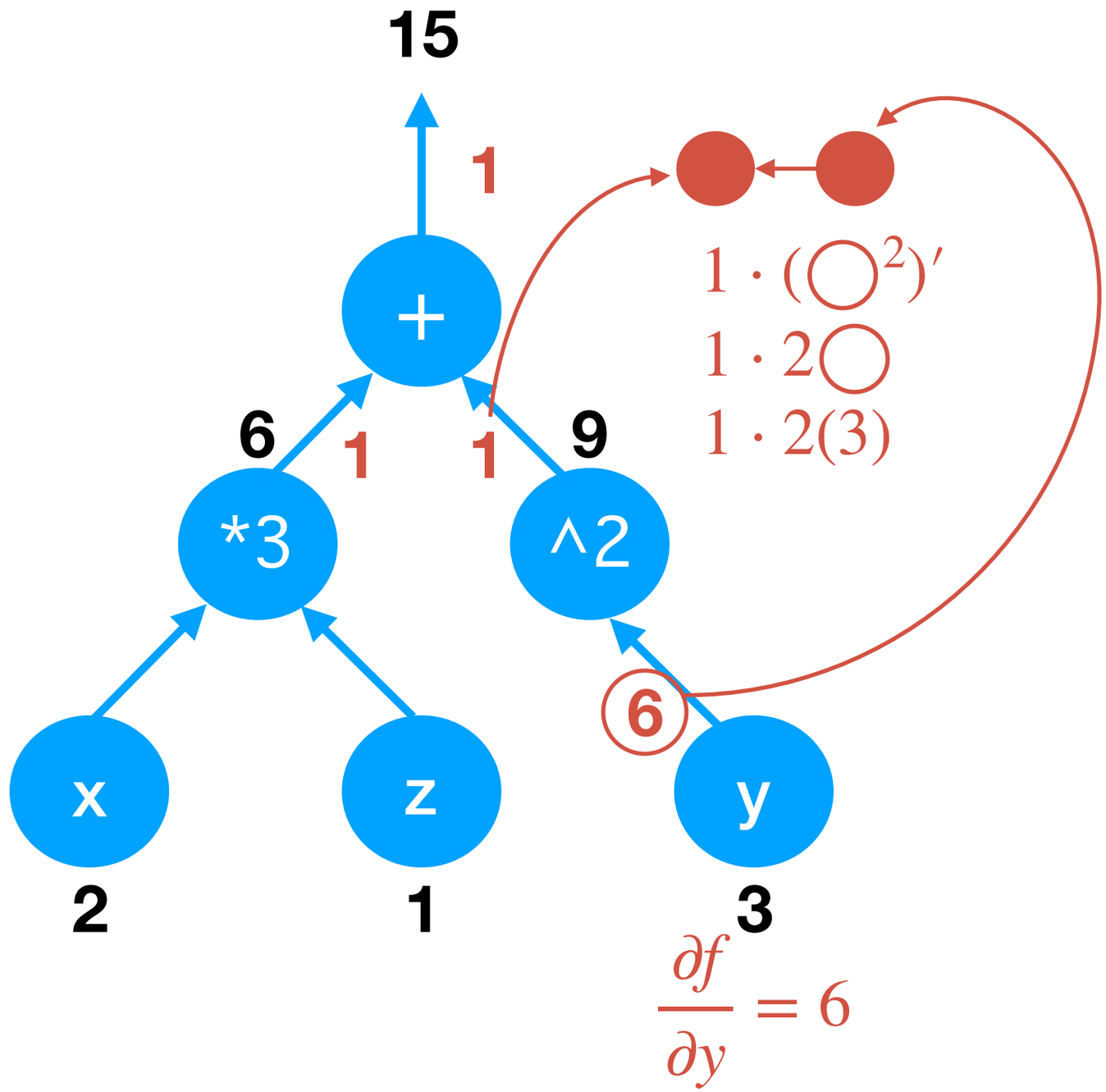
$$(f \circ g)' = (f' \circ g) \cdot g'$$

Encapsulating Function Gradient

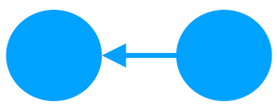
Node Gradient

Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$
$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:



$$(f \circ g)' = (f' \circ g) \cdot g'$$

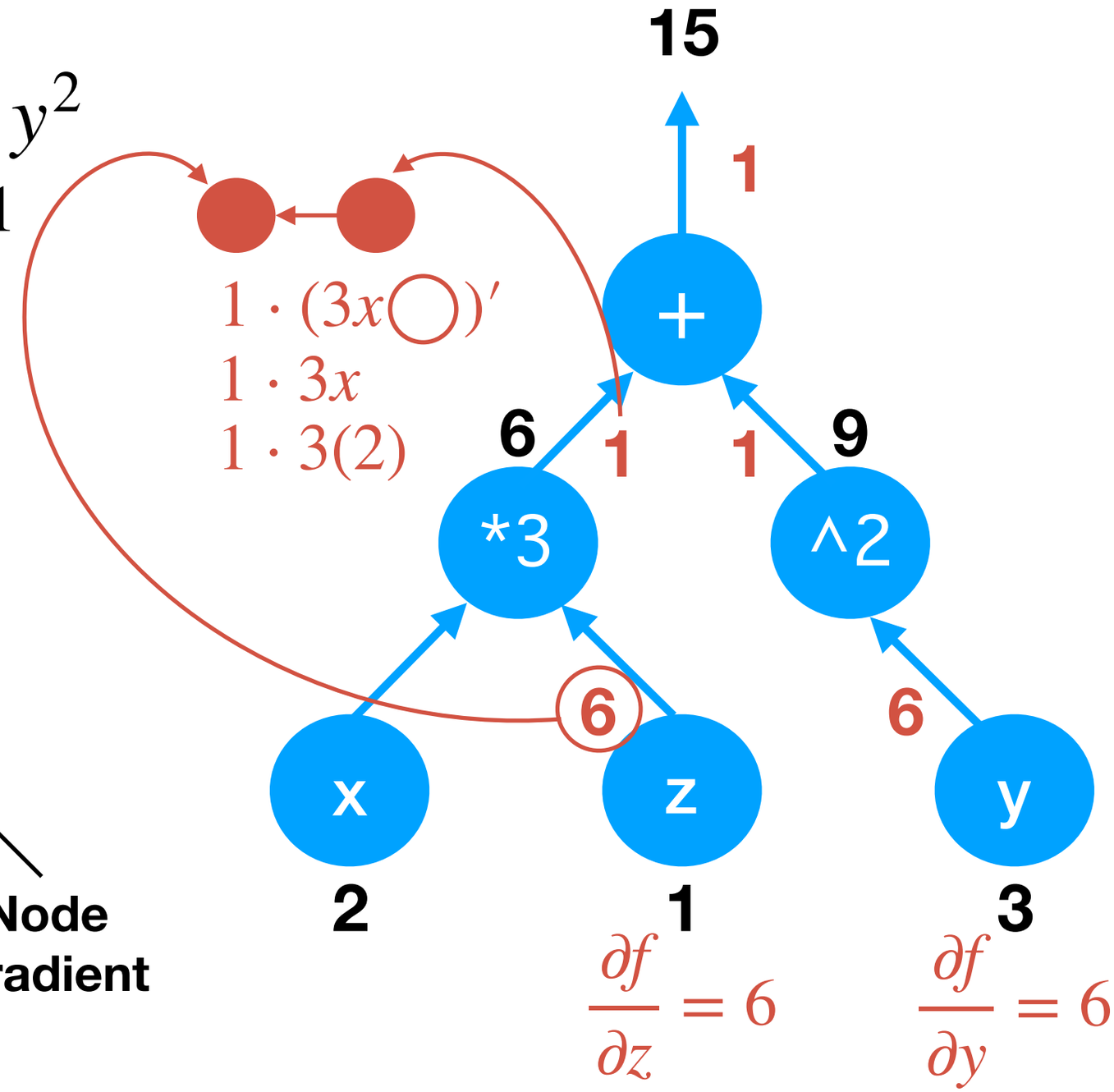
Encapsulating Function Gradient

Node Gradient

Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$

$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:

$$(f \circ g)' = (f' \circ g) \cdot g'$$

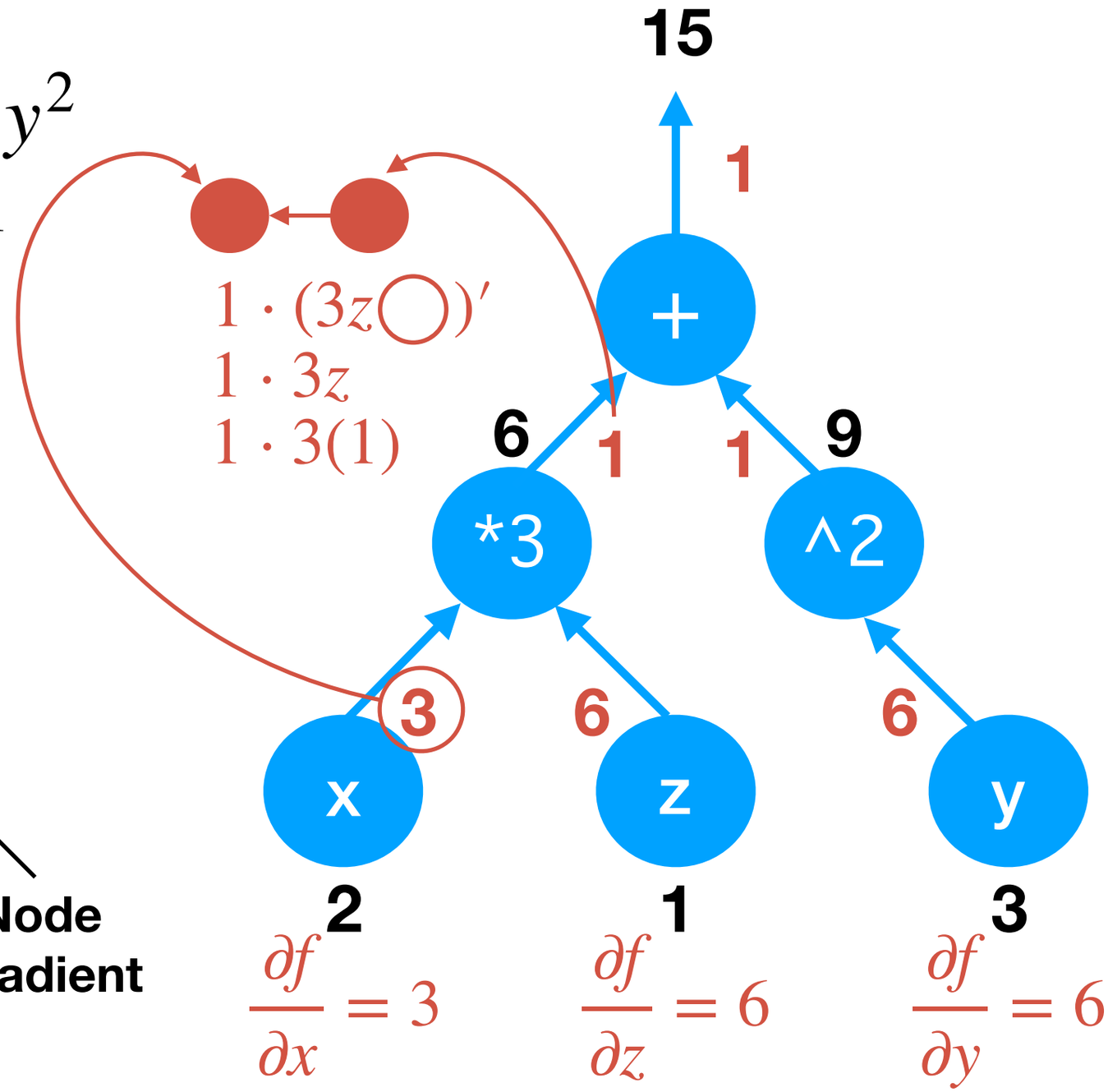
Encapsulating Function Gradient

Node Gradient

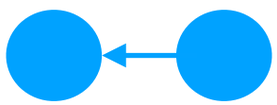
Functions as Computation Graphs

$$f(x, y, z) = 3xz + y^2$$

$$x = 2,\ y = 3,\ z = 1$$



Chain Rule:



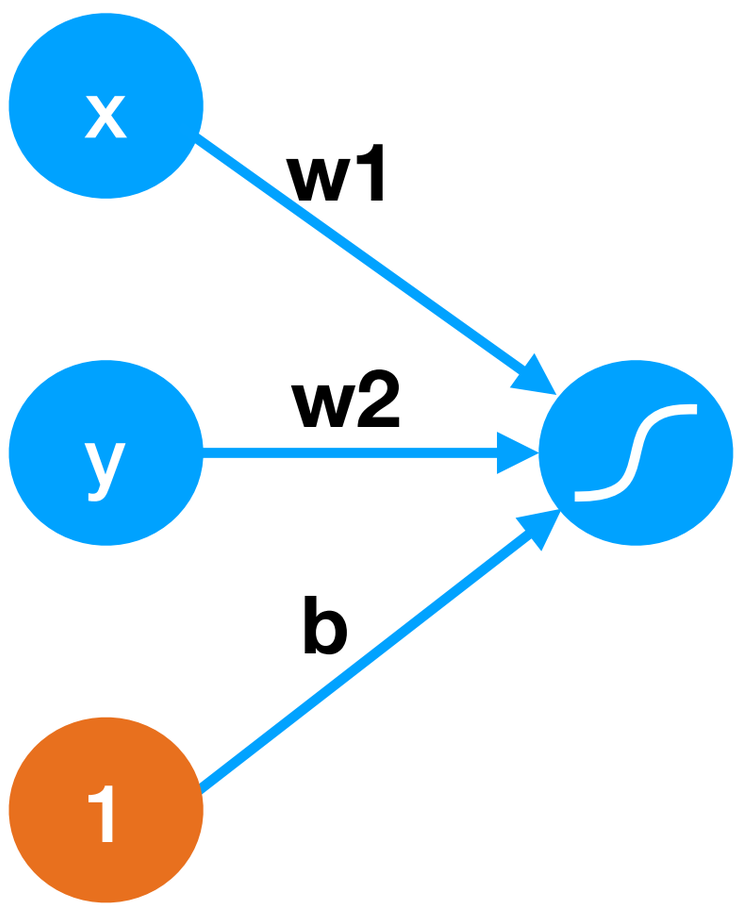
$$(f \circ g)' = (f' \circ g) \cdot g'$$

Encapsulating Function Gradient

Node Gradient

Perceptron Example

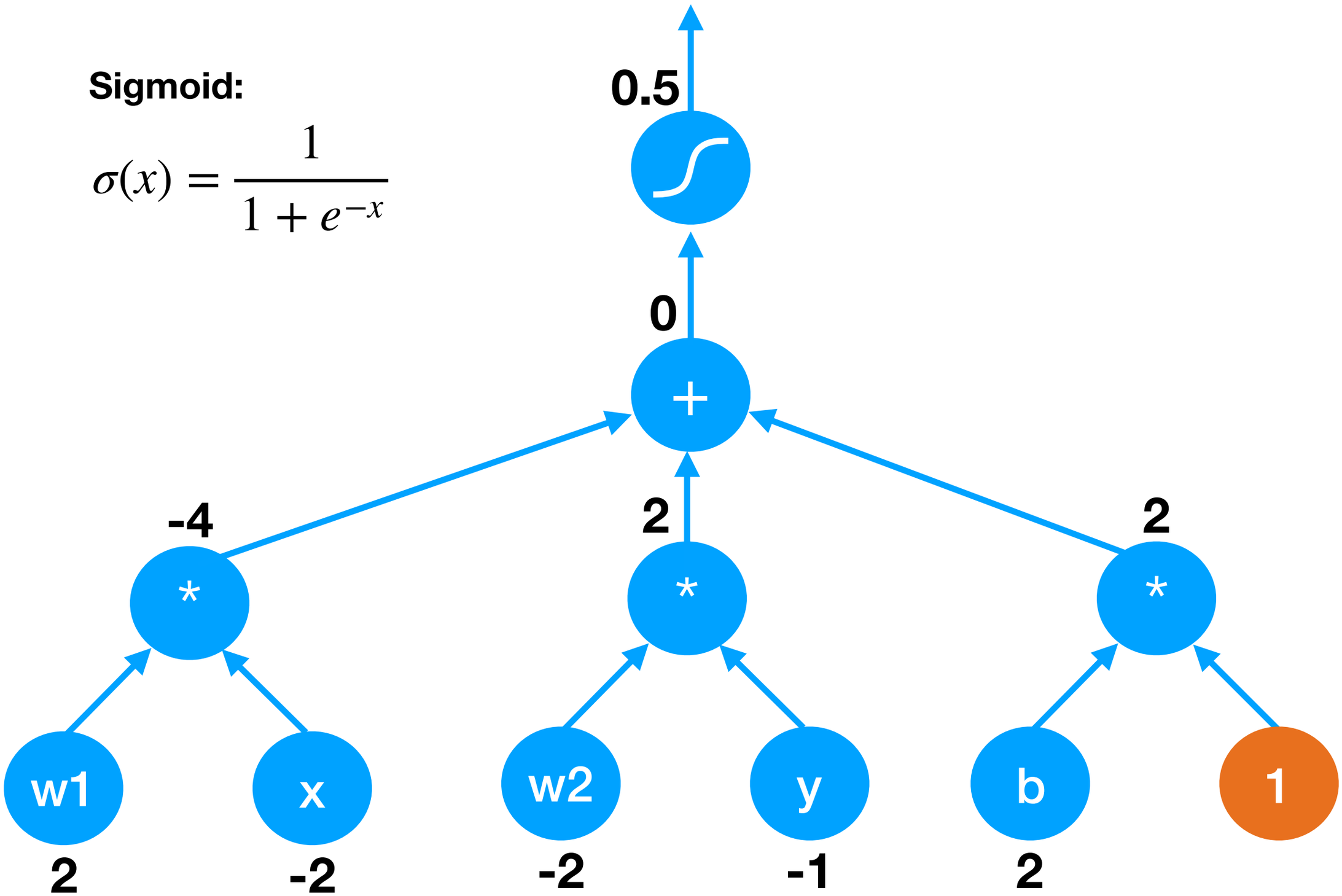
$$f(x, y) = \sigma(w_1 x + w_2 y + b)$$



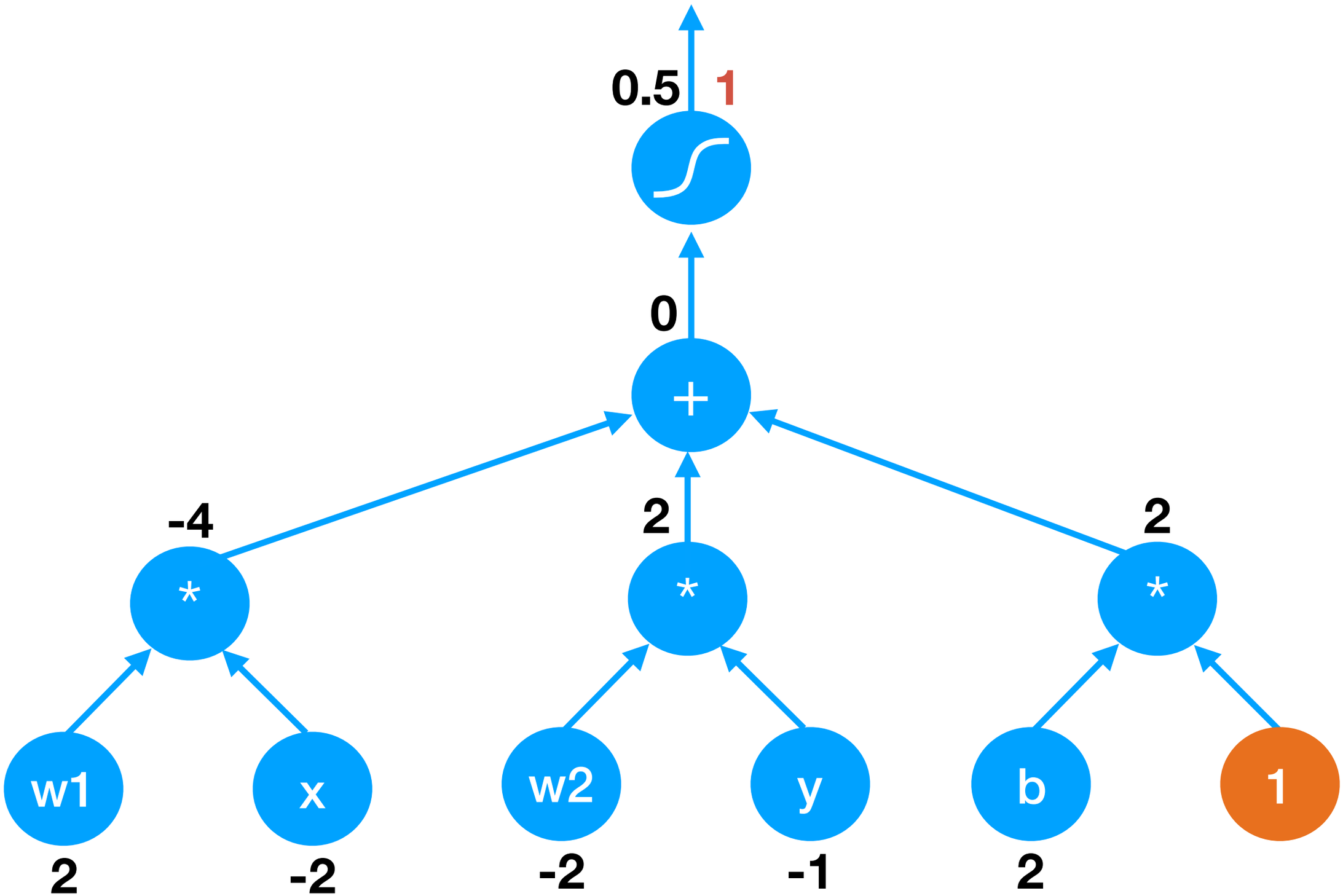
Perceptron Example

Sigmoid:

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



Perceptron Example



Perceptron Example

Sigmoid Derivative:

$$\sigma(x)' = \frac{-1}{(1+e^{-x})^2}(1+e^{-x})'$$

$$= \frac{-1}{(1+e^{-x})^2} \cdot 1 \cdot (e^{-x})'$$

$$= \frac{-1}{(1+e^{-x})^2} \cdot 1 \cdot e^{-x} \cdot (-x)'$$

$$= \frac{-1}{(1+e^{-x})^2} \cdot 1 \cdot e^{-x} \cdot -1$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}}{1+e^{-x}} + \frac{-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

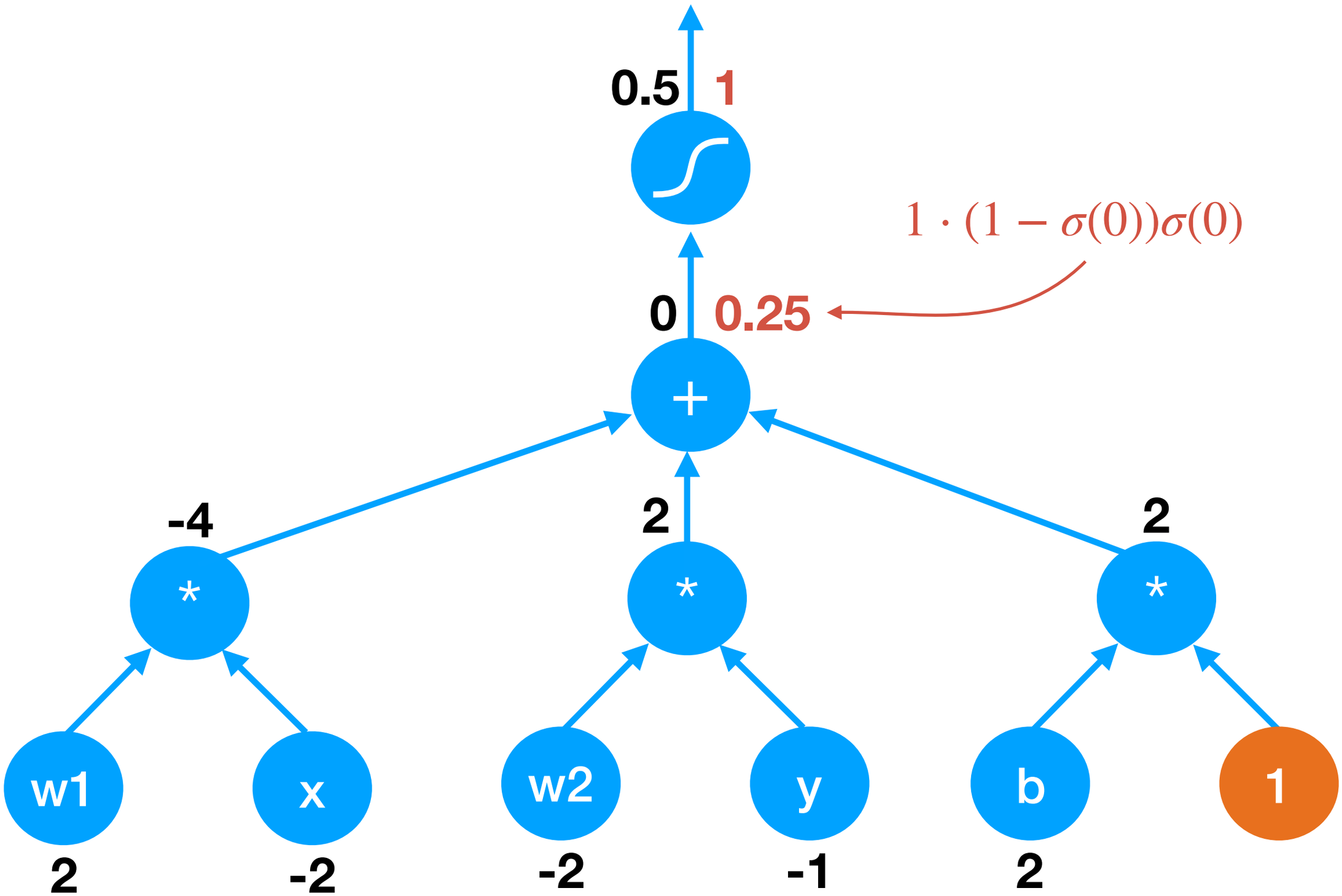
because $\left(\frac{1}{x}\right)' = -\frac{1}{x^2}$

because $(x+c)' = 1$

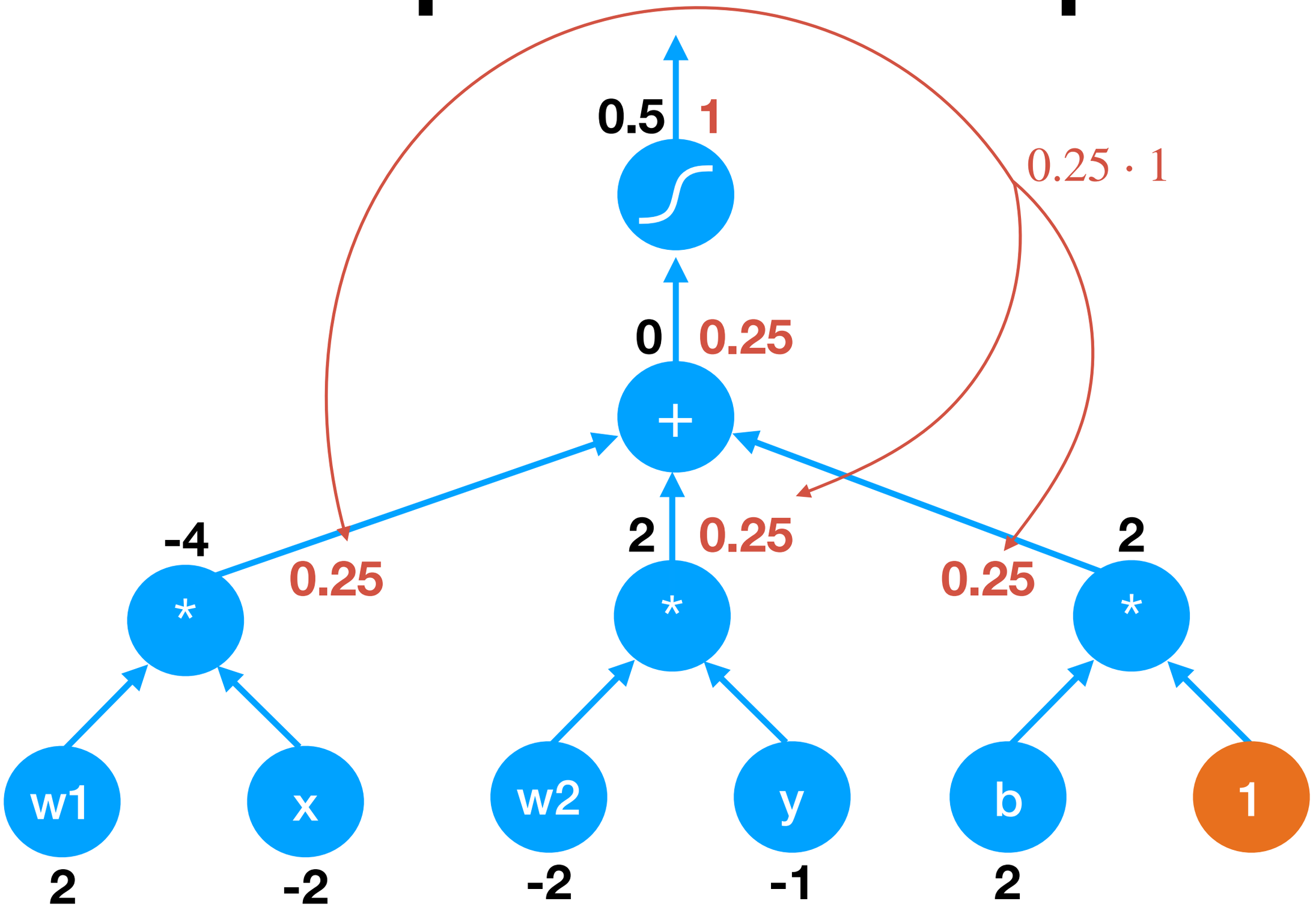
because $\left(e^x\right)' = e^x$

because $(ax)' = a$

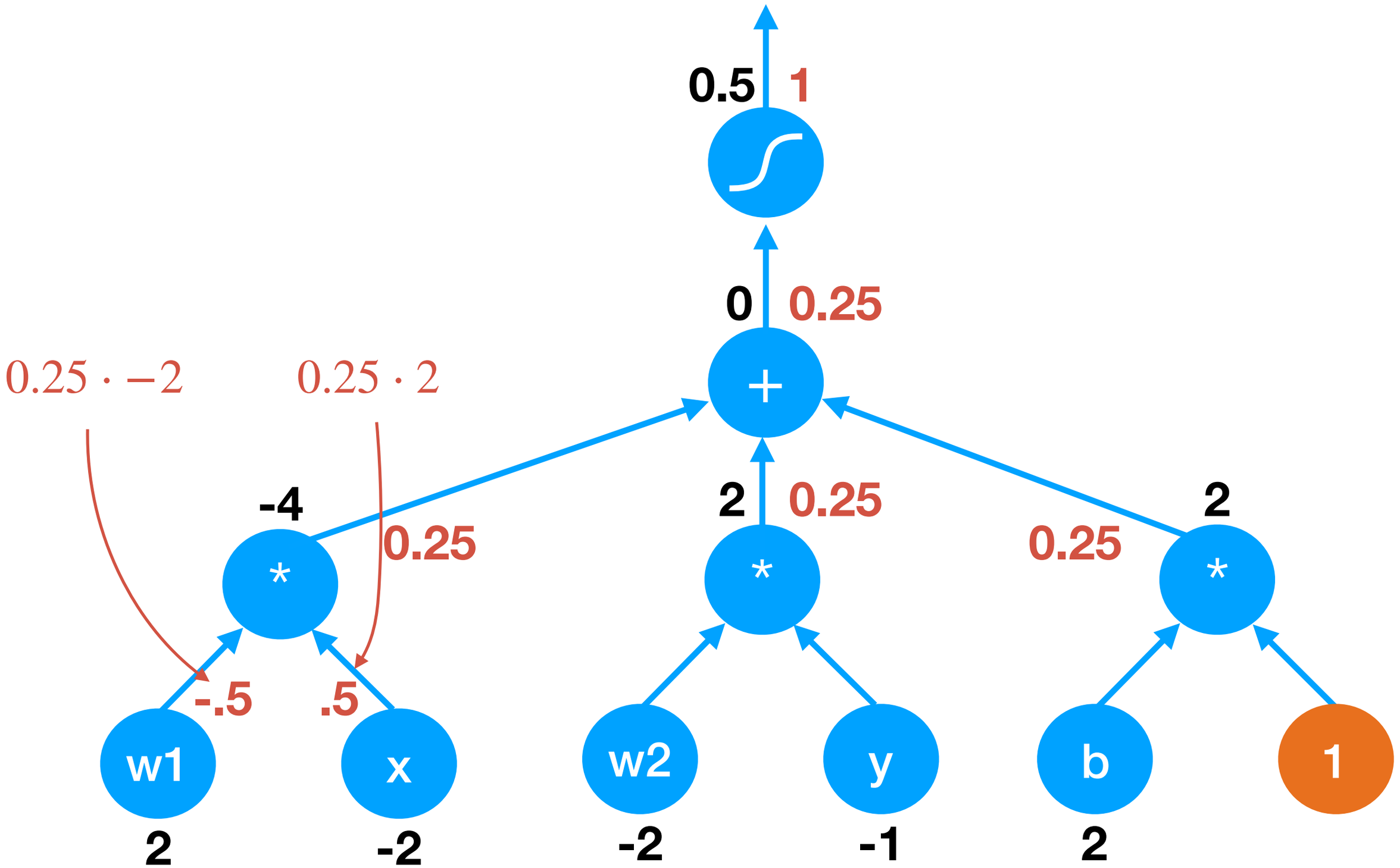
Perceptron Example



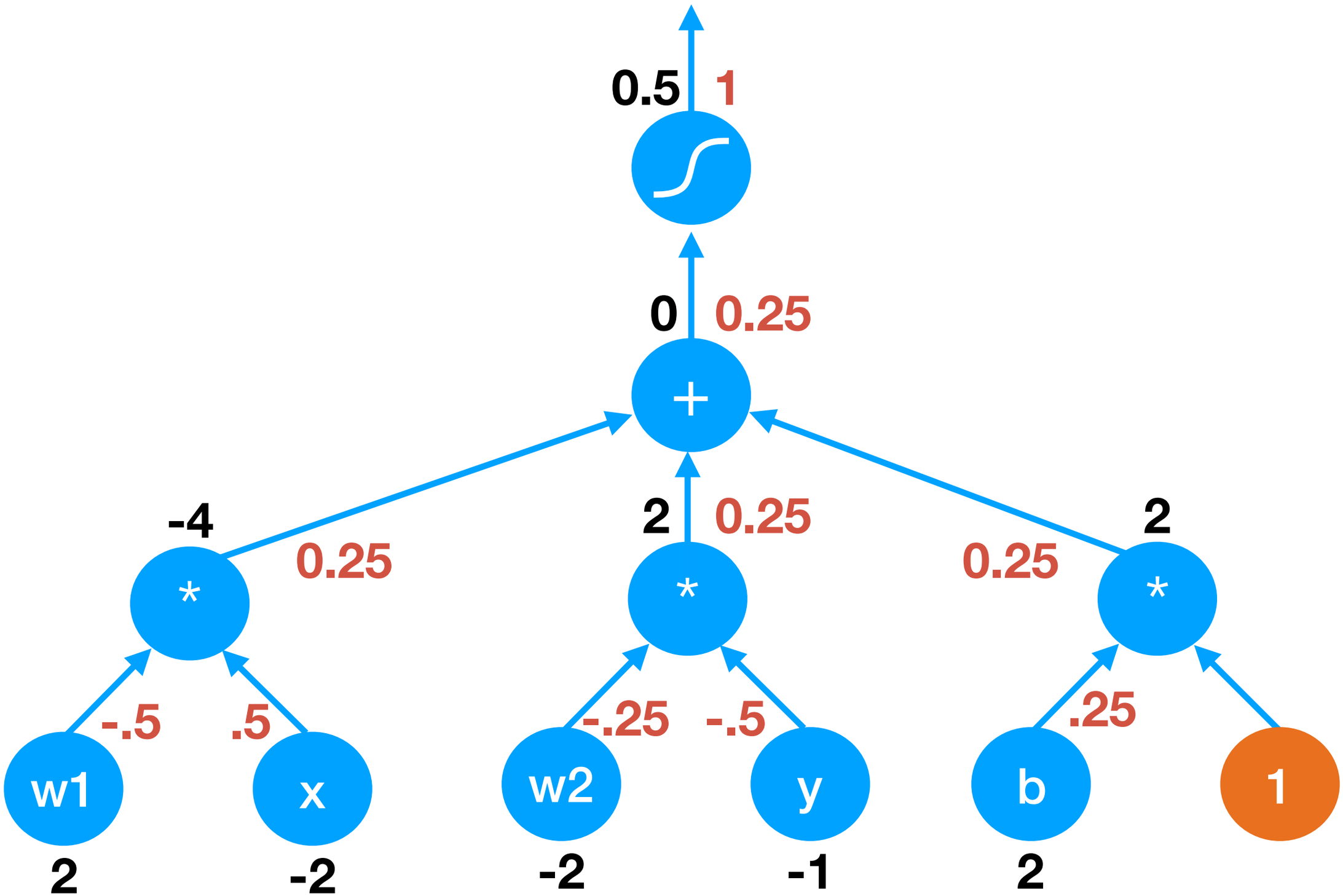
Perceptron Example



Perceptron Example



Perceptron Example

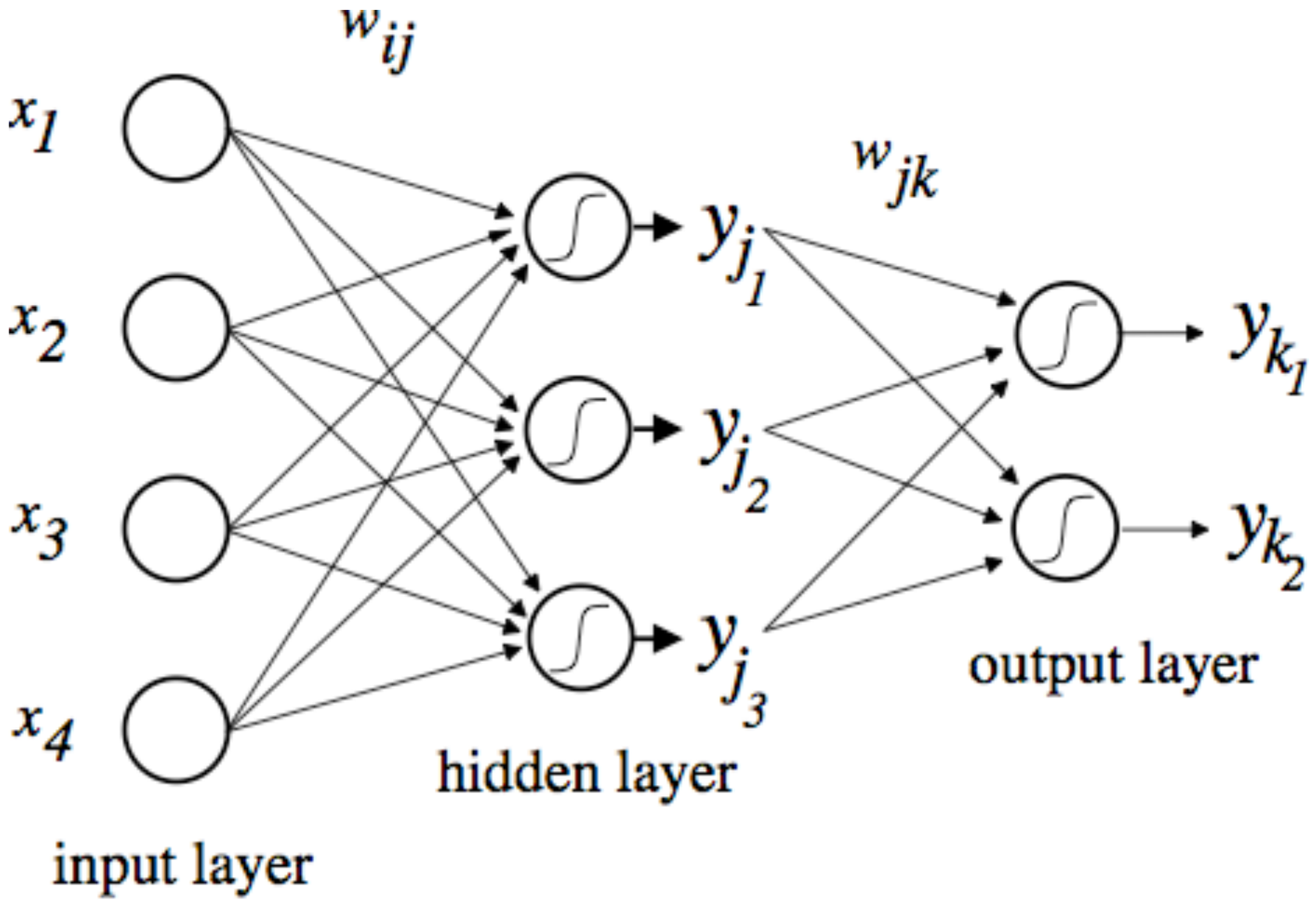


Derivatives are Cheap

- Each derivative equals the local partial derivative times the upstream derivative, thanks to the chain rule
- $+ \rightarrow$ pass through gradients
- $\times \rightarrow$ multiply other inputs by upstream gradient
- $\sigma \rightarrow$ squish gradients

Training a Neural Network

Multi-layer Perceptron (MLP)

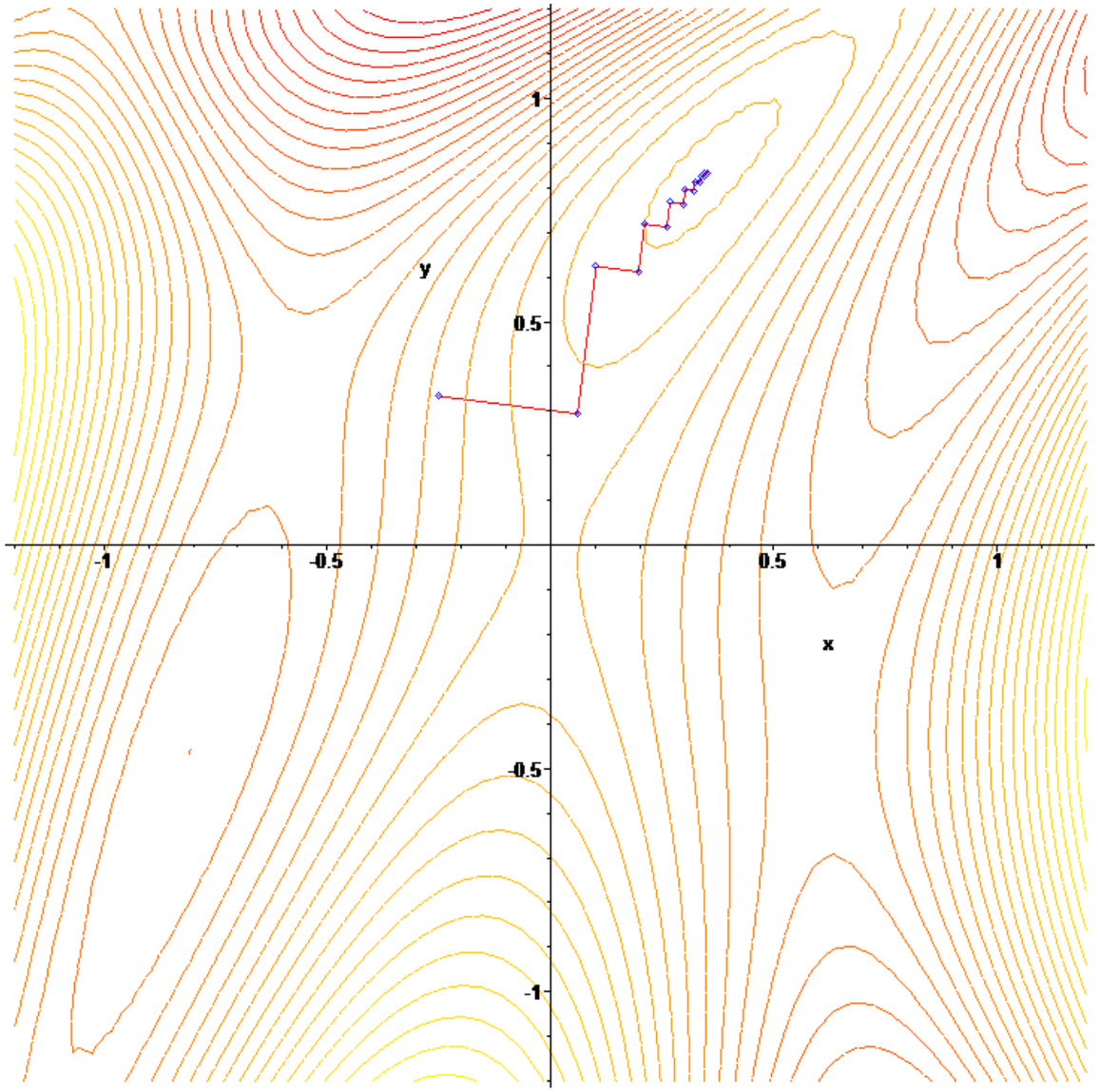


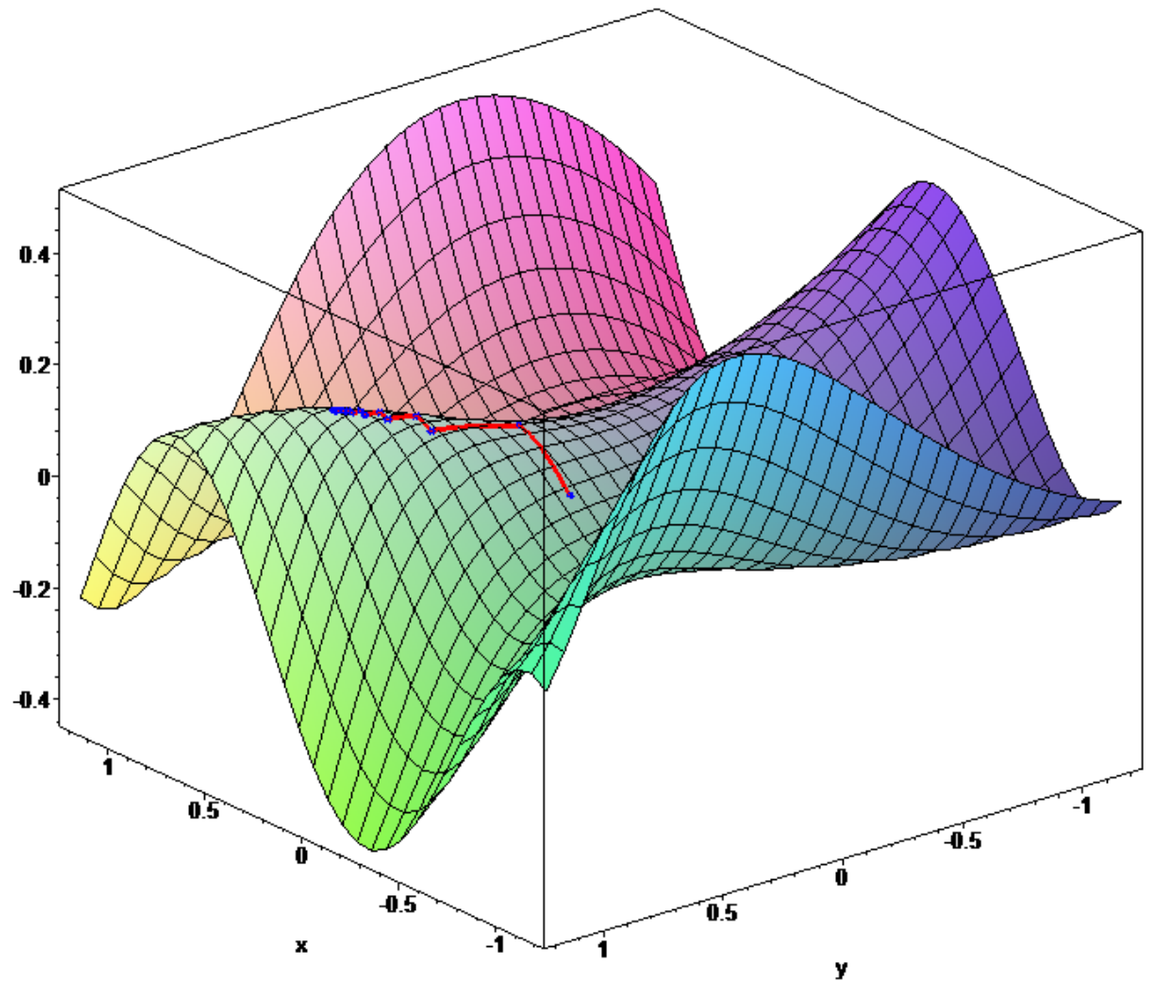
Backpropagation

- Forward Pass: present training input pattern to network and activate network to produce output (can also do in batch: present all patterns in succession)

- Backward Pass: calculate error gradient and update weights starting at output layer and then going back

Gradient Descent





Building a Neural Network

Weight Update Equation

$$\theta_w' = \theta_w - \eta * \frac{\partial TC}{\partial \theta_w}$$

New weight

Learning Rate

Gradient Descent

Input : list of n training examples $(x_0, d_0) \ldots . (x_n, d_n)$
where $\forall i : d_i \in \{+1, -1\}$

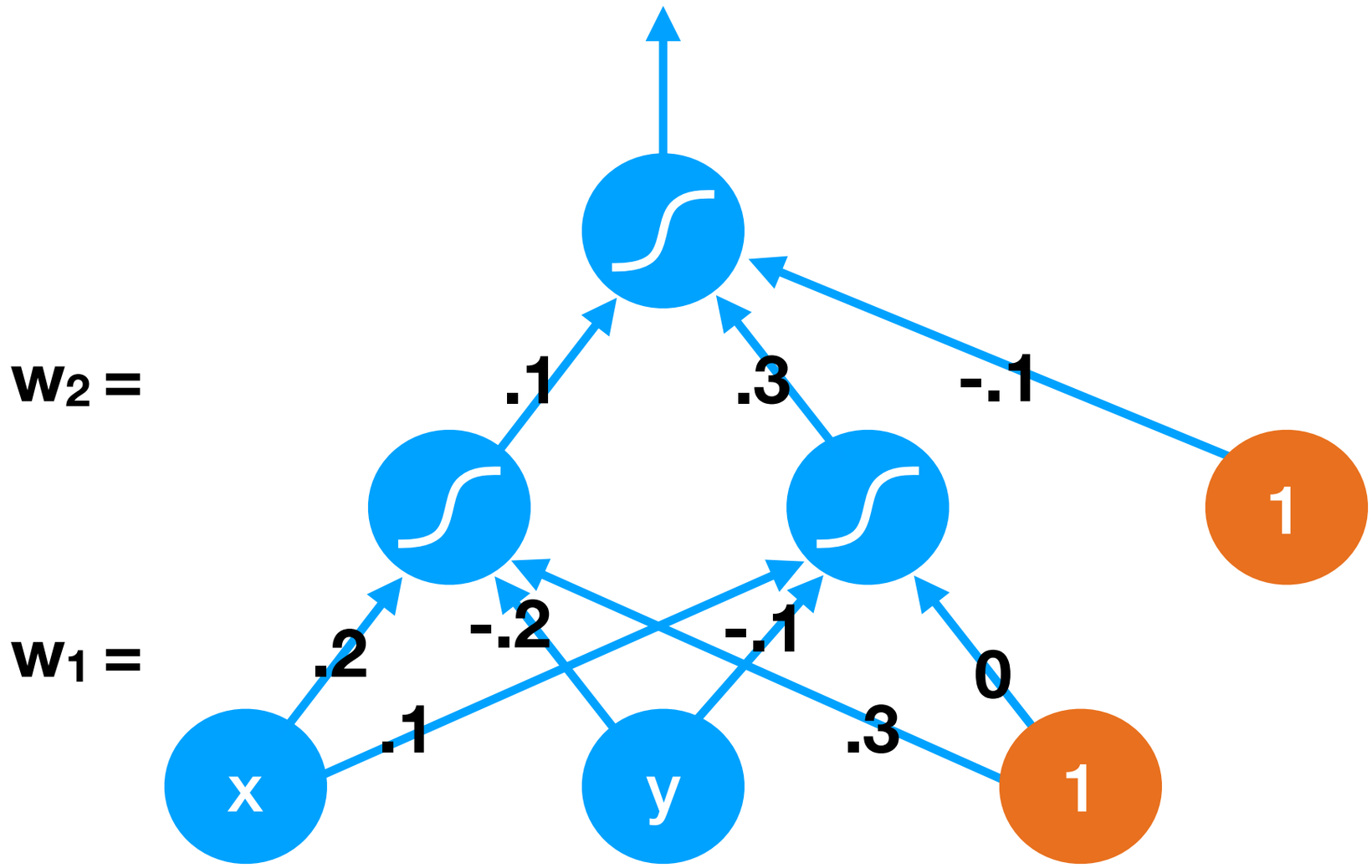
Output : classifying hyperplane w

Algorithm :

Randomly initialize w;
While makes errors on training set **do**
  for $(x_i\, d_i)$ **do**
    let $y_i = \text{MLP}(w, x_i)$;
    $loss \leftarrow \text{Mean}((d_i - y_i)^2)$
    $w' \leftarrow \text{Backprop}(w, loss)$
    $w \leftarrow w - \eta w'$
  end
end

x and w are vectors;
i is the instance index

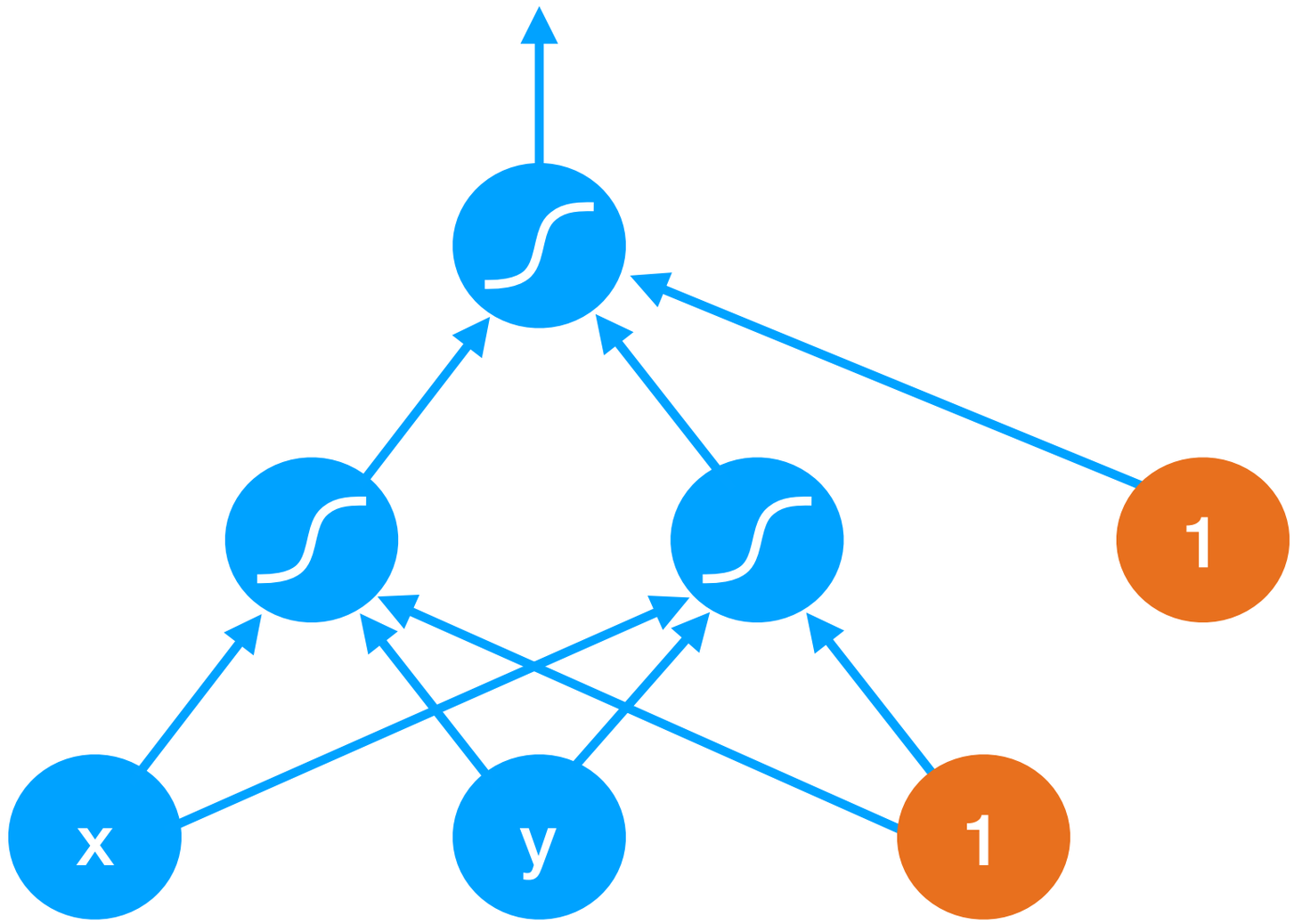
MLP Example



MLP Example

$$w_2 = \begin{bmatrix} .1 \\ .3 \\ -.1 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} .2 & .1 \\ -.2 & -.1 \\ .3 & 0 \end{bmatrix}$$



MLP Example

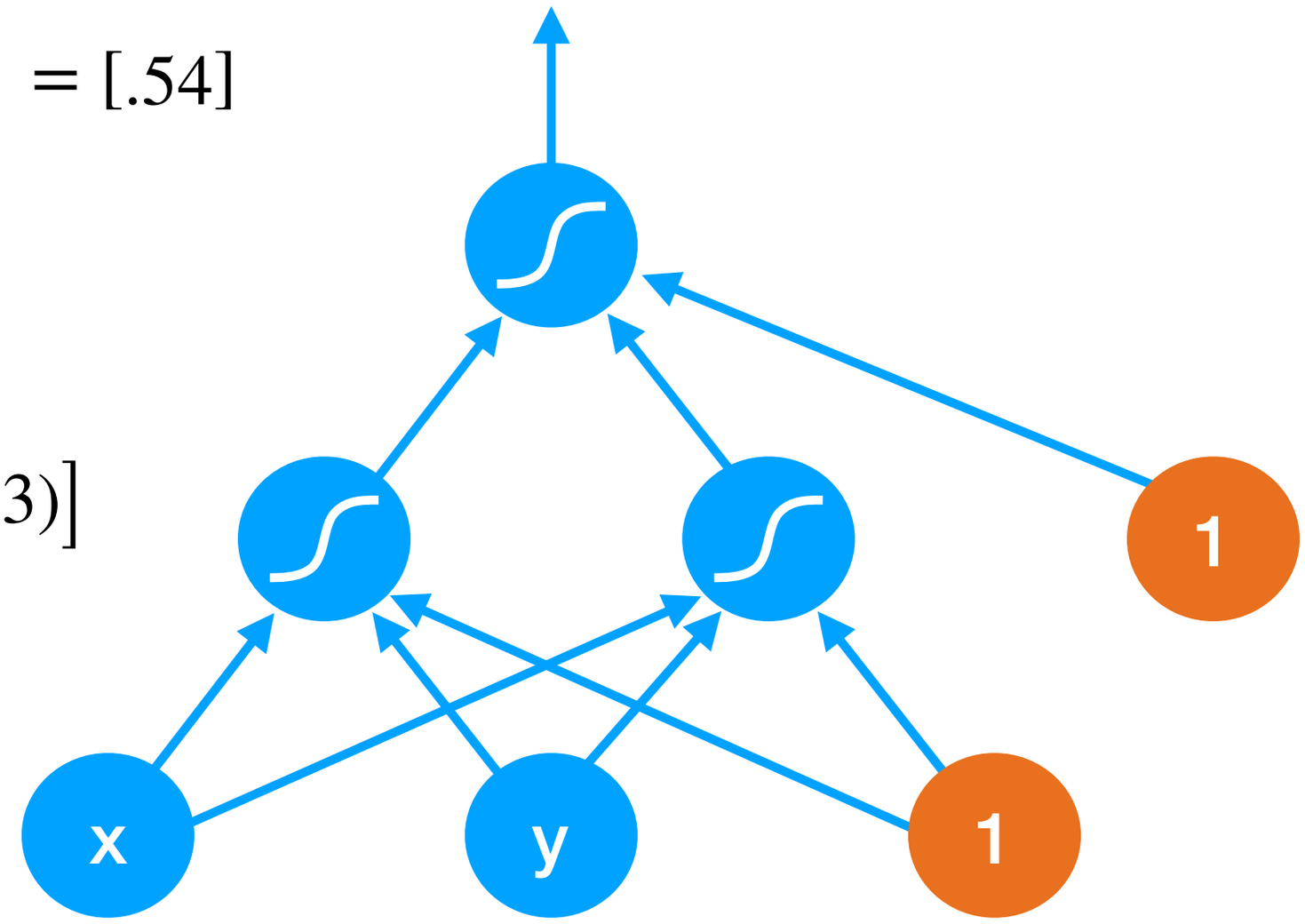
$$f(x) = \sigma(h \times w_2) = \left[\sigma(.14)\right] = [.54]$$

$$w_2 = \begin{bmatrix} .1 \\ .3 \\ -.1 \end{bmatrix}$$

$$h = \sigma(x \times w_1) = \begin{bmatrix} \sigma(.9) & \sigma(.3) \end{bmatrix}$$

$$w_1 = \begin{bmatrix} .2 & .1 \\ -.2 & -.1 \\ .3 & 0 \end{bmatrix}$$

$$x = [2 \quad -1 \quad 1]$$



Calculate Gradients

$$f(x) = \sigma(h \times w_2) = \left[\sigma(.14)\right] = [.54]$$

$$w_2 = \begin{bmatrix} .1 \\ .3 \\ -.1 \end{bmatrix}$$

$$h = \sigma(x \times w_1) = \begin{bmatrix} \sigma(.9) & \sigma(.3) \end{bmatrix}$$

$$w_1 = \begin{bmatrix} .2 & .1 \\ -.2 & -.1 \\ .3 & 0 \end{bmatrix}$$

$$x = [2 \quad -1 \quad 1]$$

Target = 1 → $\text{Loss} = (1 - .54)^2 = .21$

Loss Gradient → $-2 \cdot .46 = -.92$

Sigmoid Gradient → $-.92 \cdot (1 - \sigma(.14))\sigma(.14) = -.23$

$$w_2' = -.23 \begin{bmatrix} \sigma(.9) \\ \sigma(.3) \\ 1 \end{bmatrix} = \begin{bmatrix} -.16 \\ -.13 \\ -.23 \end{bmatrix}$$

$$h' = -.23 \begin{bmatrix} .1 & .3 \end{bmatrix} = [-.02 \quad -.07]$$

Sigmoid Gradient →

$$[-.02 \cdot (1 - \sigma(.9))\sigma(.9), -.07 \cdot (1 - \sigma(.3))\sigma(.3)]$$

$$[-.02, -.004]$$

$$w_1' = \begin{bmatrix} -.02 \cdot 2 & -.004 \cdot 2 \\ -.02 \cdot -1 & -.004 \cdot -1 \\ -.02 \cdot 1 & -.004 \cdot 1 \end{bmatrix} = \begin{bmatrix} -.04 & -.008 \\ .02 & .004 \\ -.02 & -.004 \end{bmatrix}$$

Update Weights

let $\eta = 0.001$

$$w_1 = w_1 - \eta w_1'$$

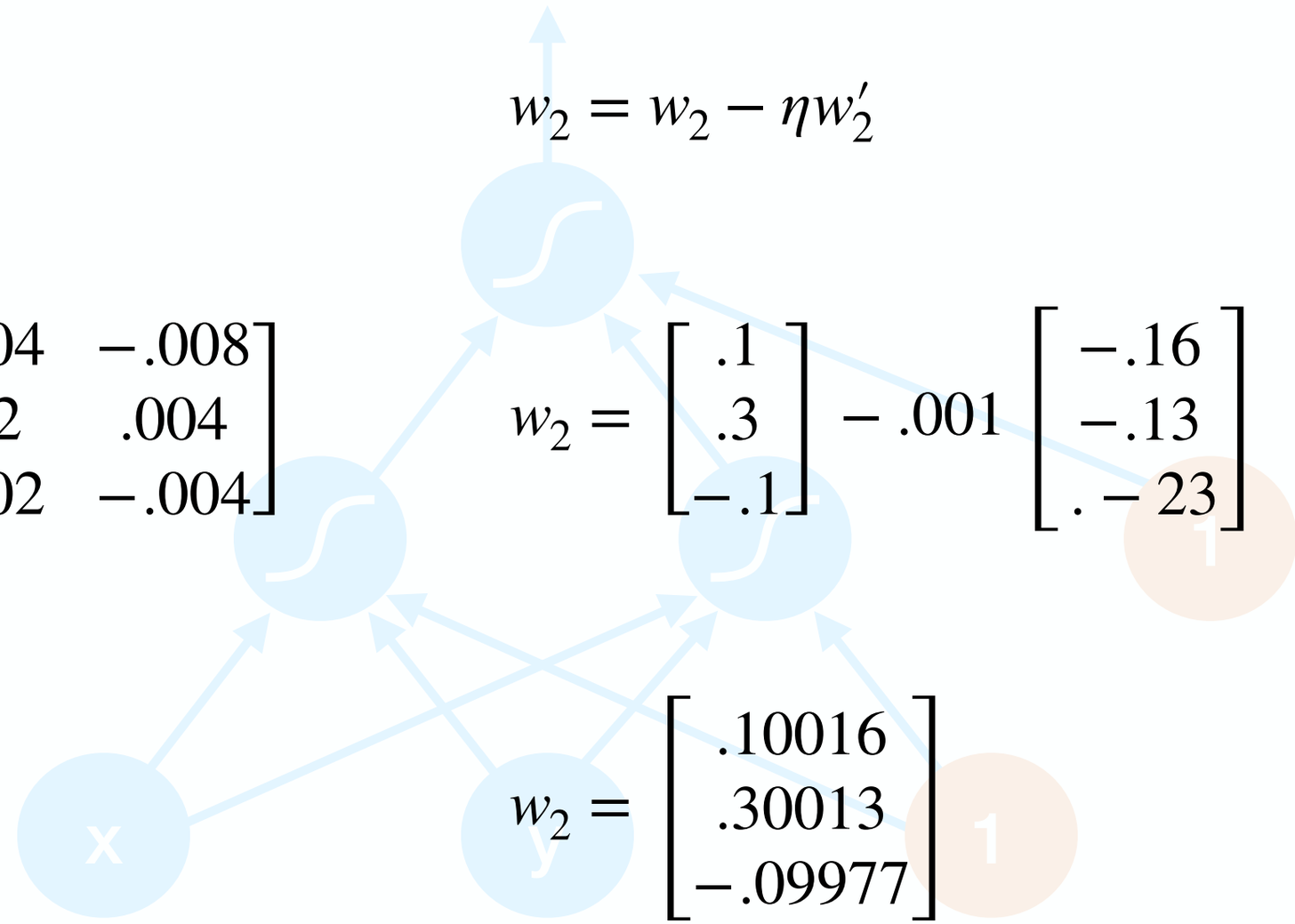
$$w_1 = \begin{bmatrix} .2 & .1 \\ -.2 & -.1 \\ .3 & 0 \end{bmatrix} - .001 \begin{bmatrix} -.04 & -.008 \\ .02 & .004 \\ -.02 & -.004 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} .20004 & .10001 \\ -.20002 & -.1000 \\ .30002 & .00000 \end{bmatrix}$$

$$w_2 = w_2 - \eta w_2'$$

$$w_2 = \begin{bmatrix} .1 \\ .3 \\ -.1 \end{bmatrix} - .001 \begin{bmatrix} -.16 \\ -.13 \\ .-23 \end{bmatrix}$$

$$w_2 = \begin{bmatrix} .10016 \\ .30013 \\ -.09977 \end{bmatrix}$$



values are rounded

Fully Connected Layer

Class FCLayer:

var $w \leftarrow$ InitializeRandom(input size + 1, output size)

var $b \leftarrow$ InitializeRandom(output size)

var $\eta \leftarrow$ Learning Rate

def forward(x):

return $x \times w + b$

def backward($grads$):

$w' \leftarrow x^{T} \times grads$

$x' \leftarrow grads \times w^{T}$

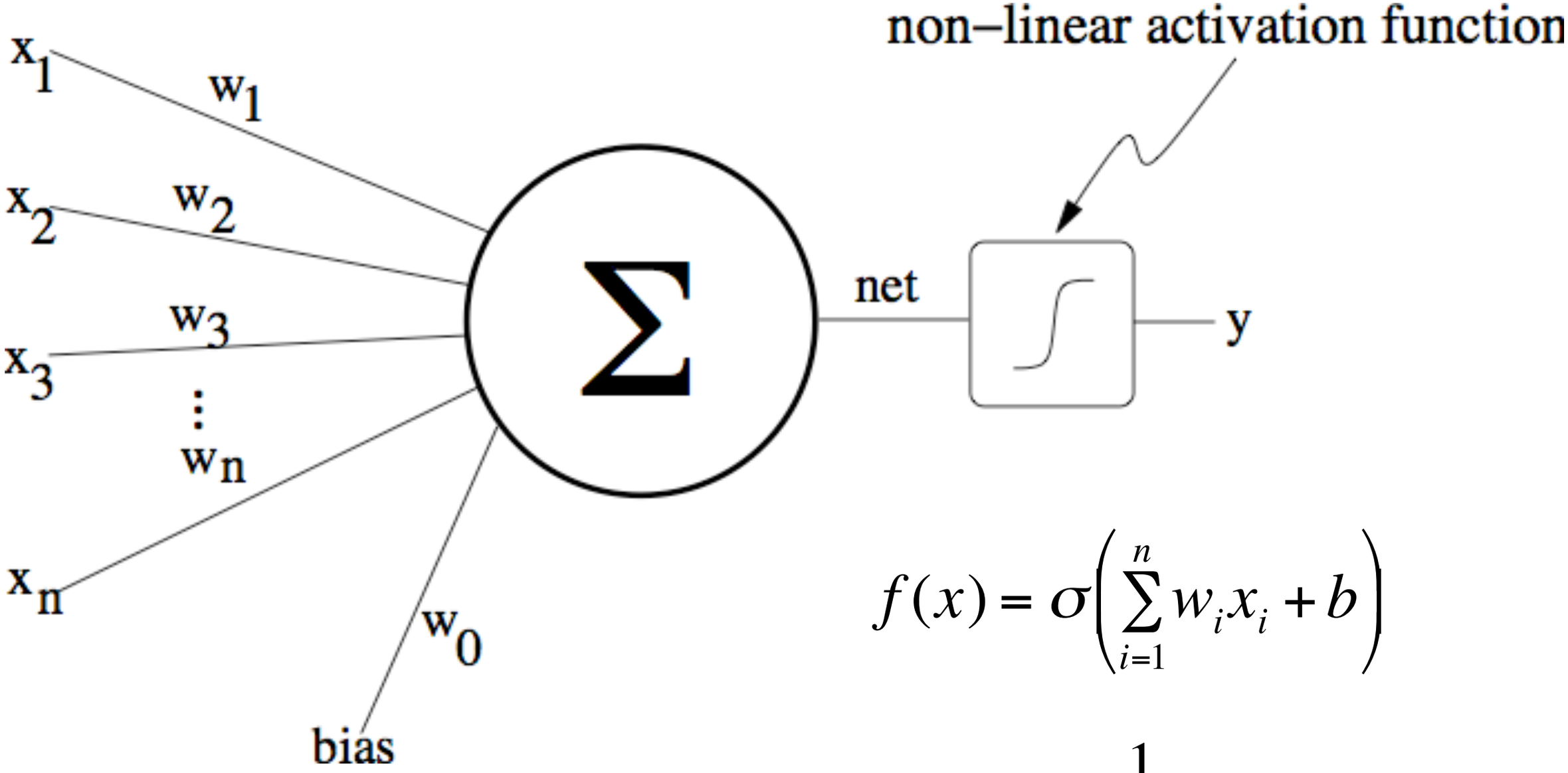
$w \leftarrow w - \eta w'$

$b \leftarrow b - \eta \cdot grads$

return x'

Just need to define an Activation Layer then you can stack them

Activation Functions



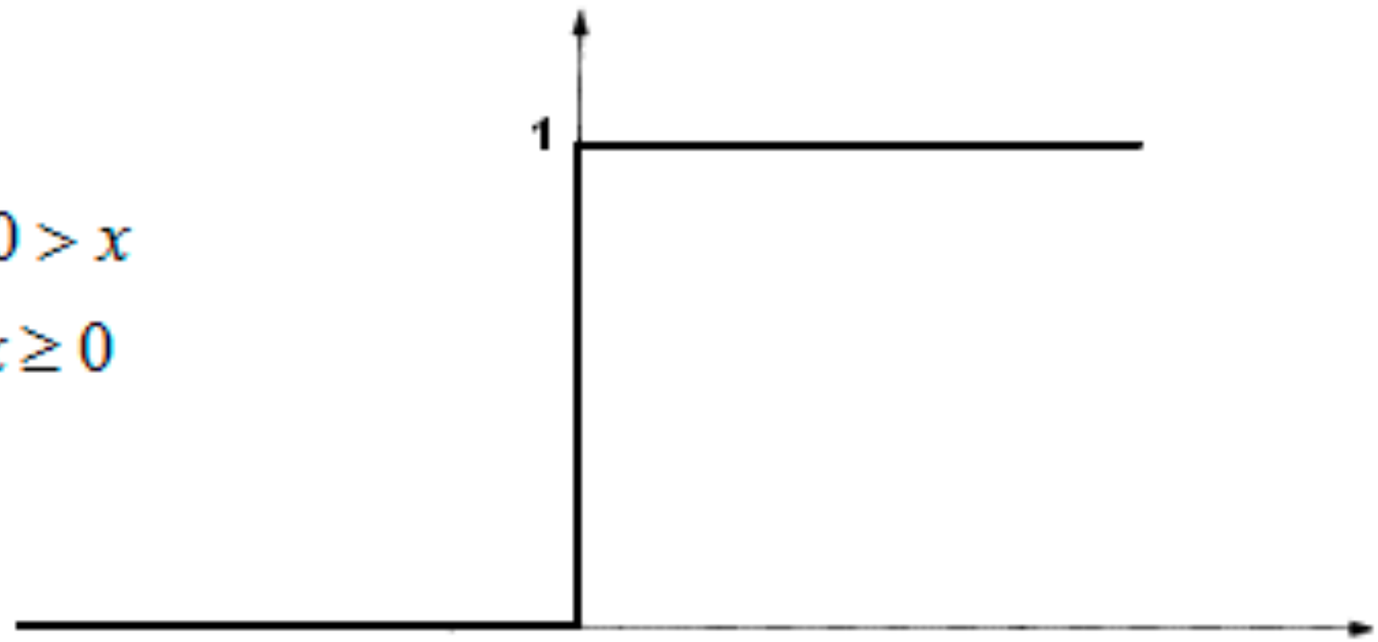
$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

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

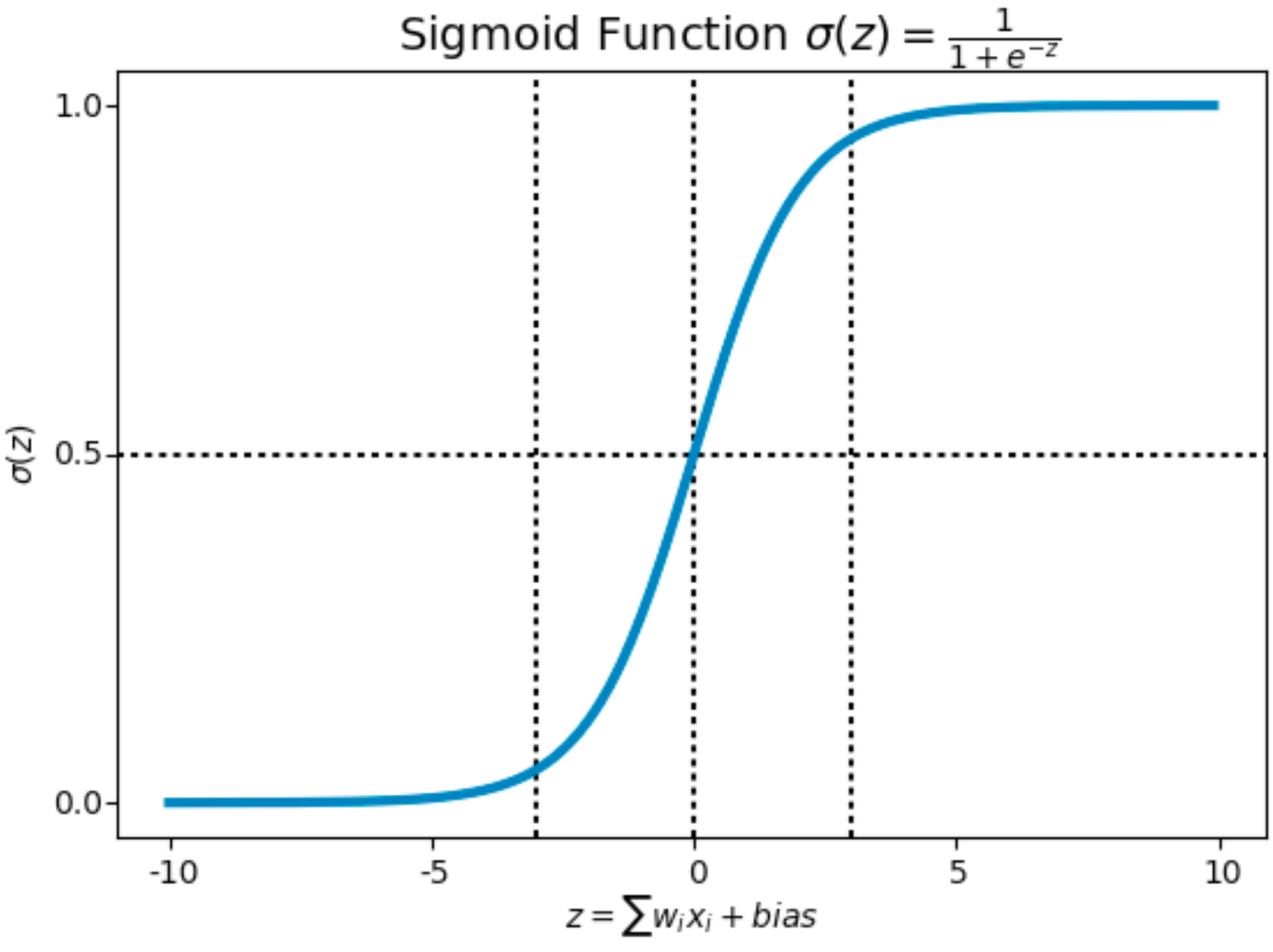
Activation Functions

Unit step (threshold)

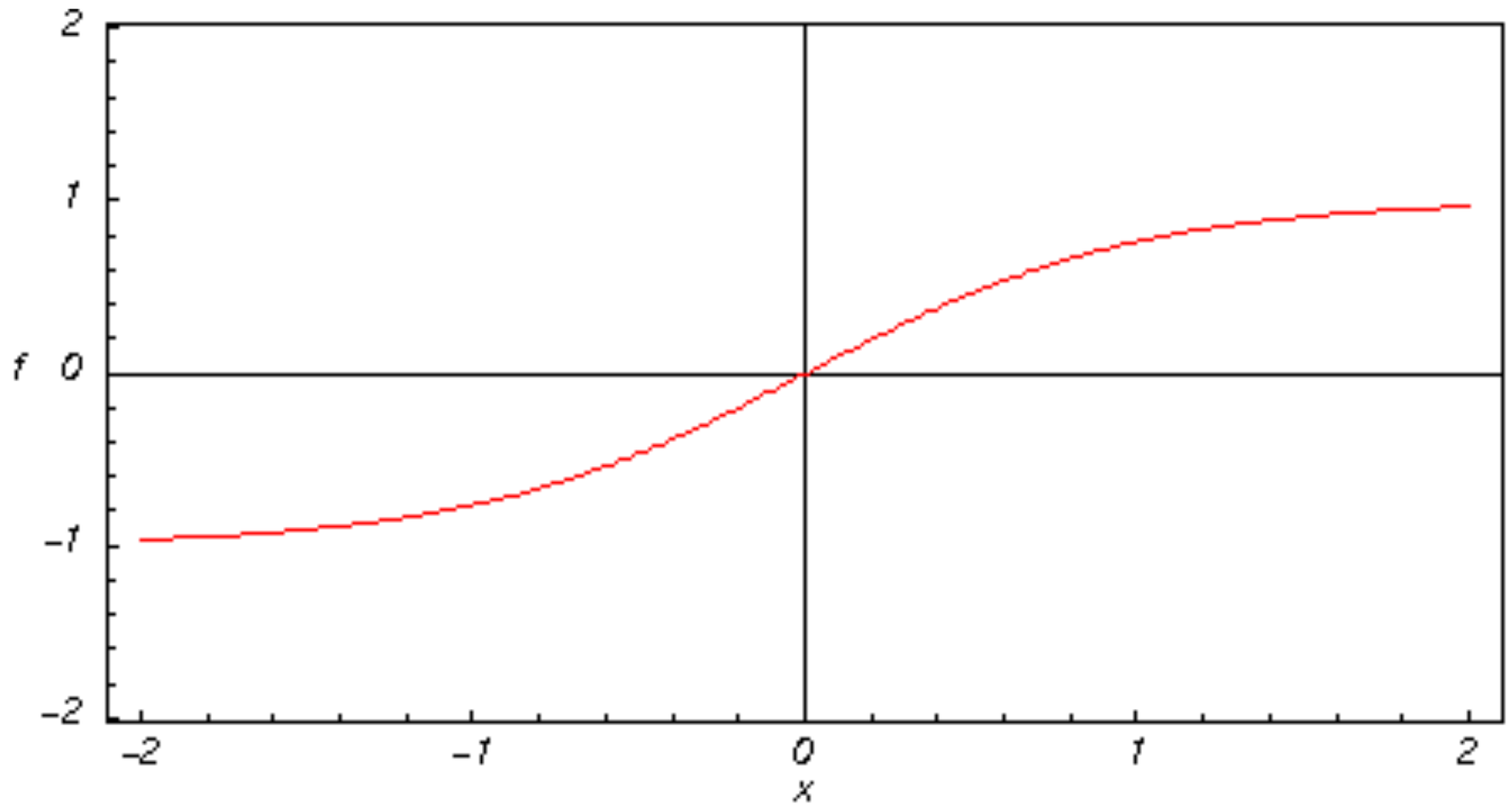
$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Activation Functions

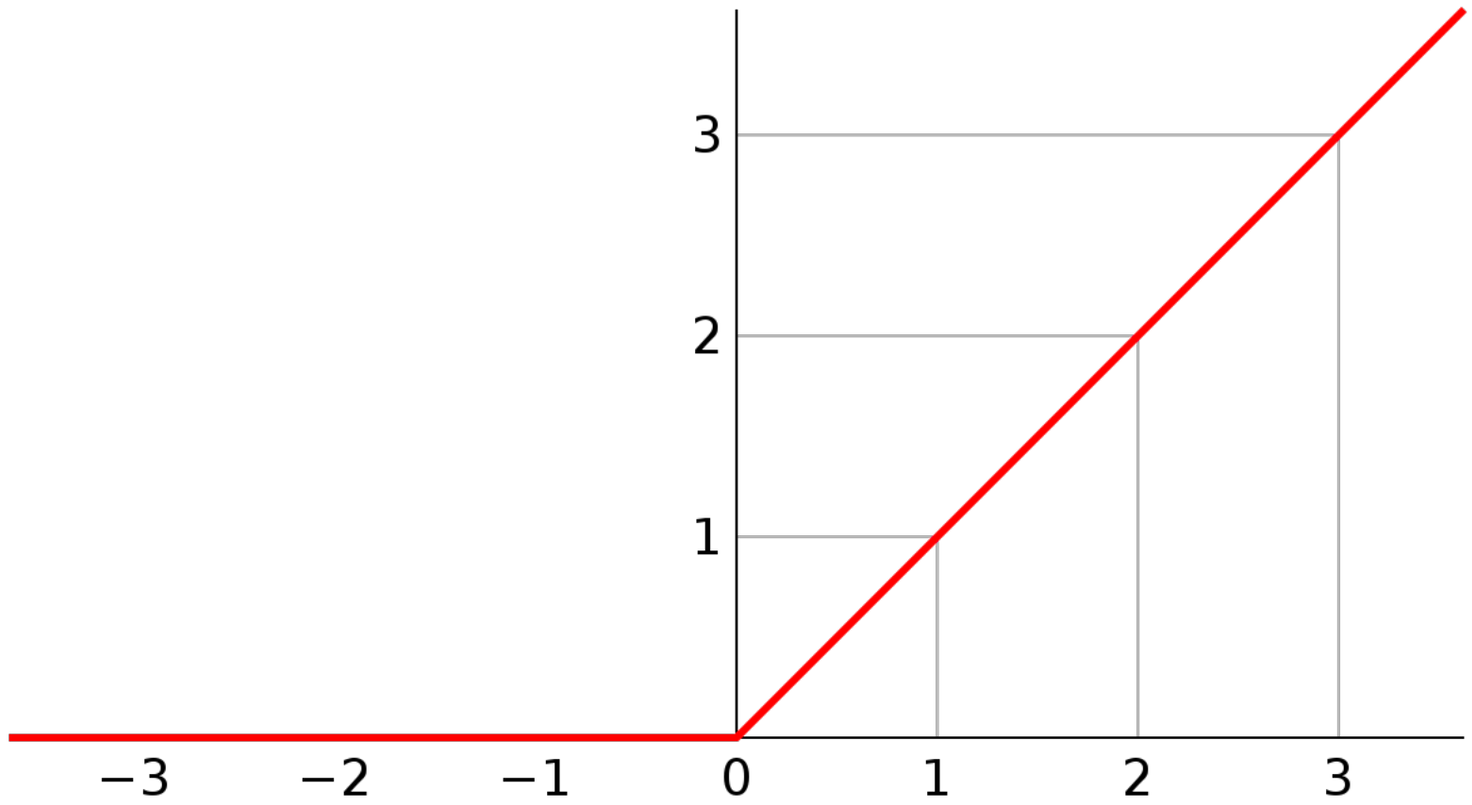


Activation Functions



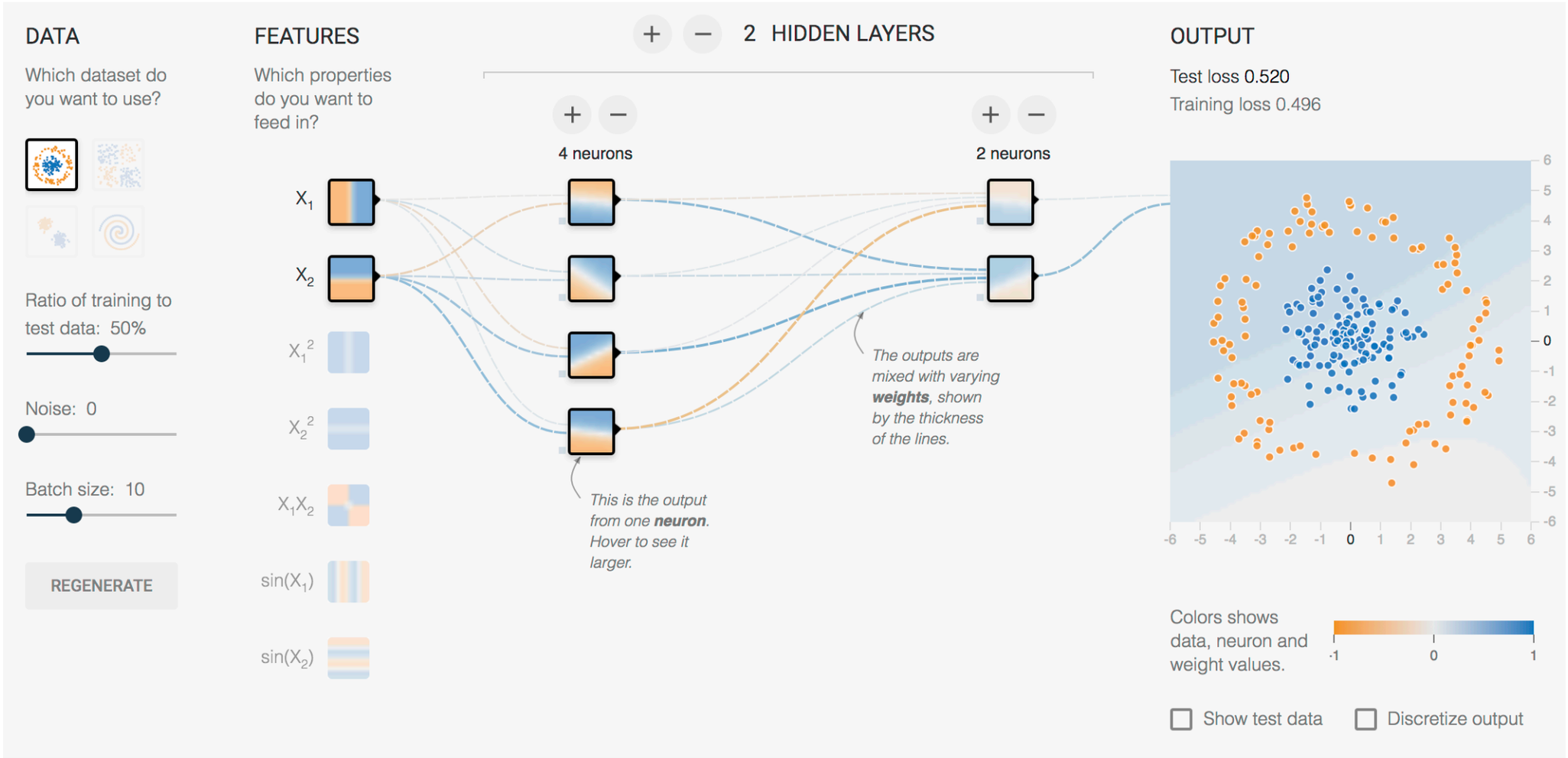
Hyperbolic Tangent

Activation Functions



Rectified Linear Units
(ReLu)

Elements of a Neural Network



http://playground.tensorflow.org/

Deep Learning

Why Deep Learning is so successful

- Deep Part (deep hierarchy improves optimization terrain)
  - Architecture, Activation, Initialization
- GPU + Internet (Data)
  - Allow more layers and weights
- Only Successful Architecture
  - CNN/RNN (only grids and sequences)

Perceptron problems

- Noise: if the data isn't separable, weights might thrash

- Multiple layers allow it to work on non-linearly separable data

- Mediocre generalization: finds a "barely" separating solution

- More data and a continuous loss function greatly improve generalization

- Overtraining: test / held-out accuracy usually rises, then falls

- Large data, many layers, and batch training help