Natural Language Processing

Lecture 18

Dirk Hovy

dirk.hovy@unibocconi.it

@dirk_hovy

Bocconi

Neural Nets Everywhere

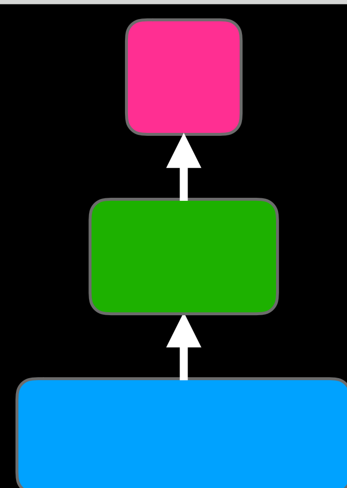
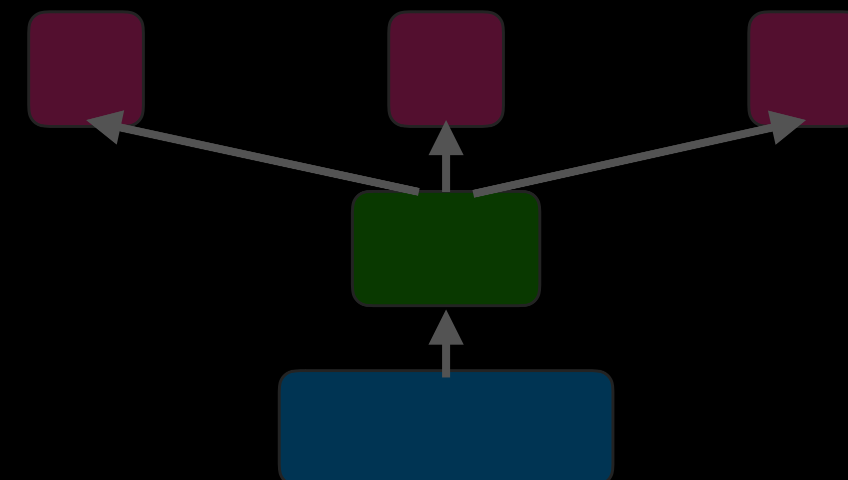
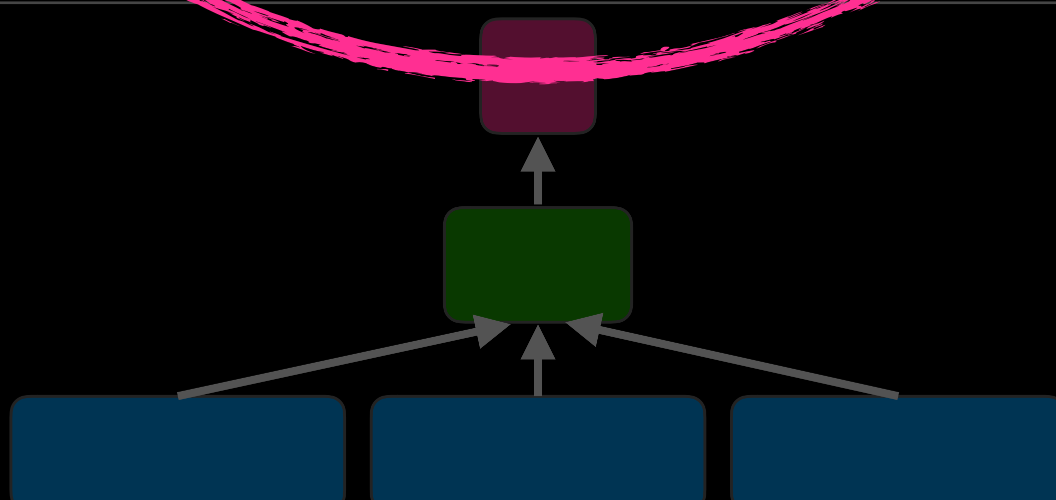
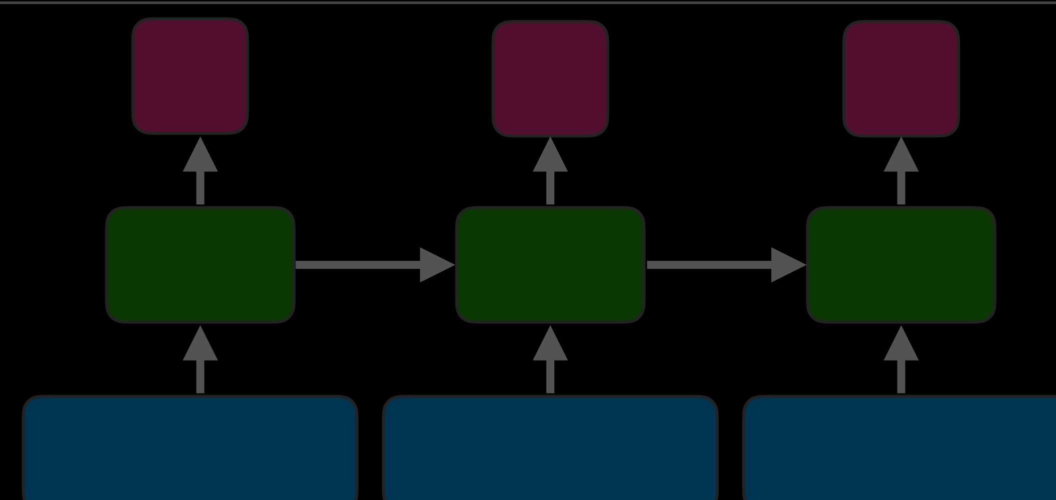
FAKE NEWS

this bird is red with white and has a very short beak

Goals for Today

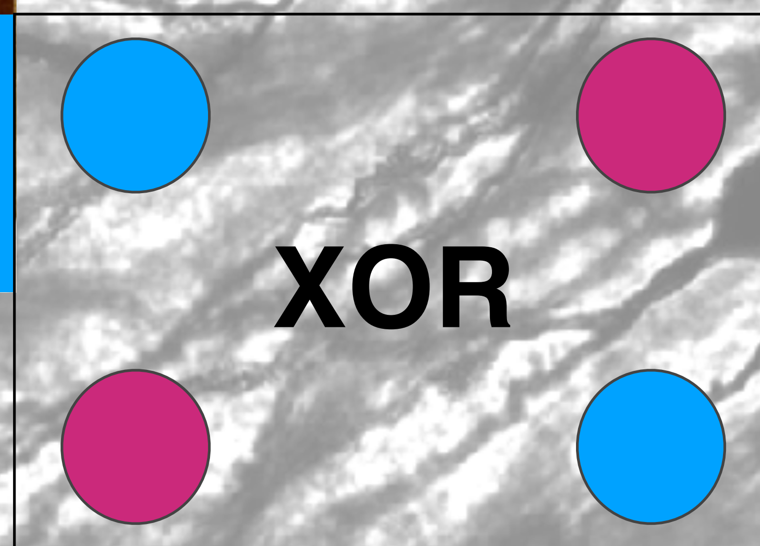
- Learn about **neural architectures**
- Understand the **perceptron** as basic element
- Understand training through **backpropagation**
- Learn about **dropout regularization**

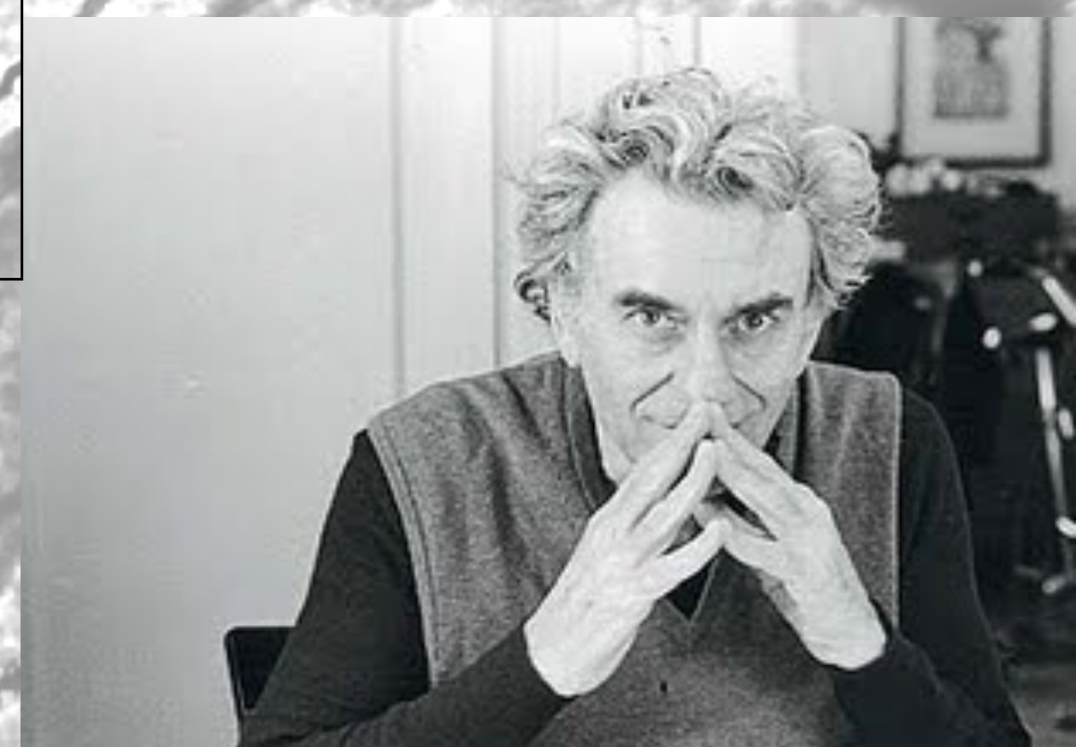
Types of Neural Models

	Fixed length	Variable length
Fixed length	 Logistic Regression, Perceptron, Feed-Forward Network, Deep Belief Network…	 Multitask Learning, Decoder
Variable length	 Convolutional Neural Networks (CNN)	 Recurrent Neural Networks (RNN), Hidden Markov Models (HMM), Conditional Random Fields

Frank Rosenblatt
(1928–1971)

The Perceptron can learn *anything!!!*



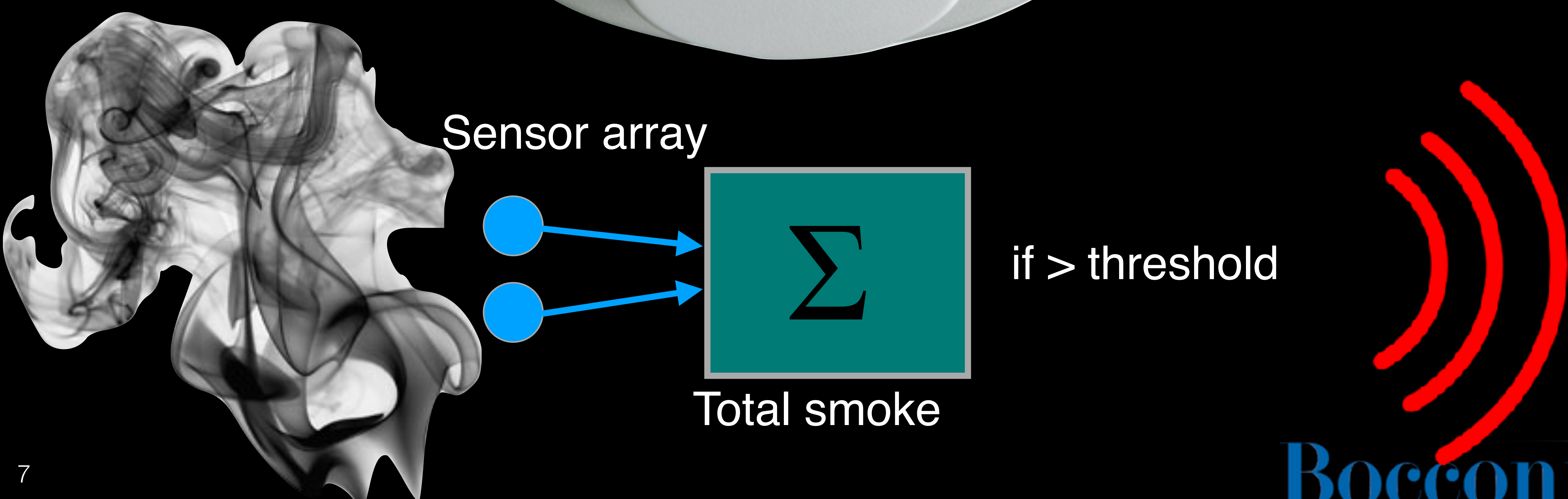


Marvin Minsky
(1927–2016)

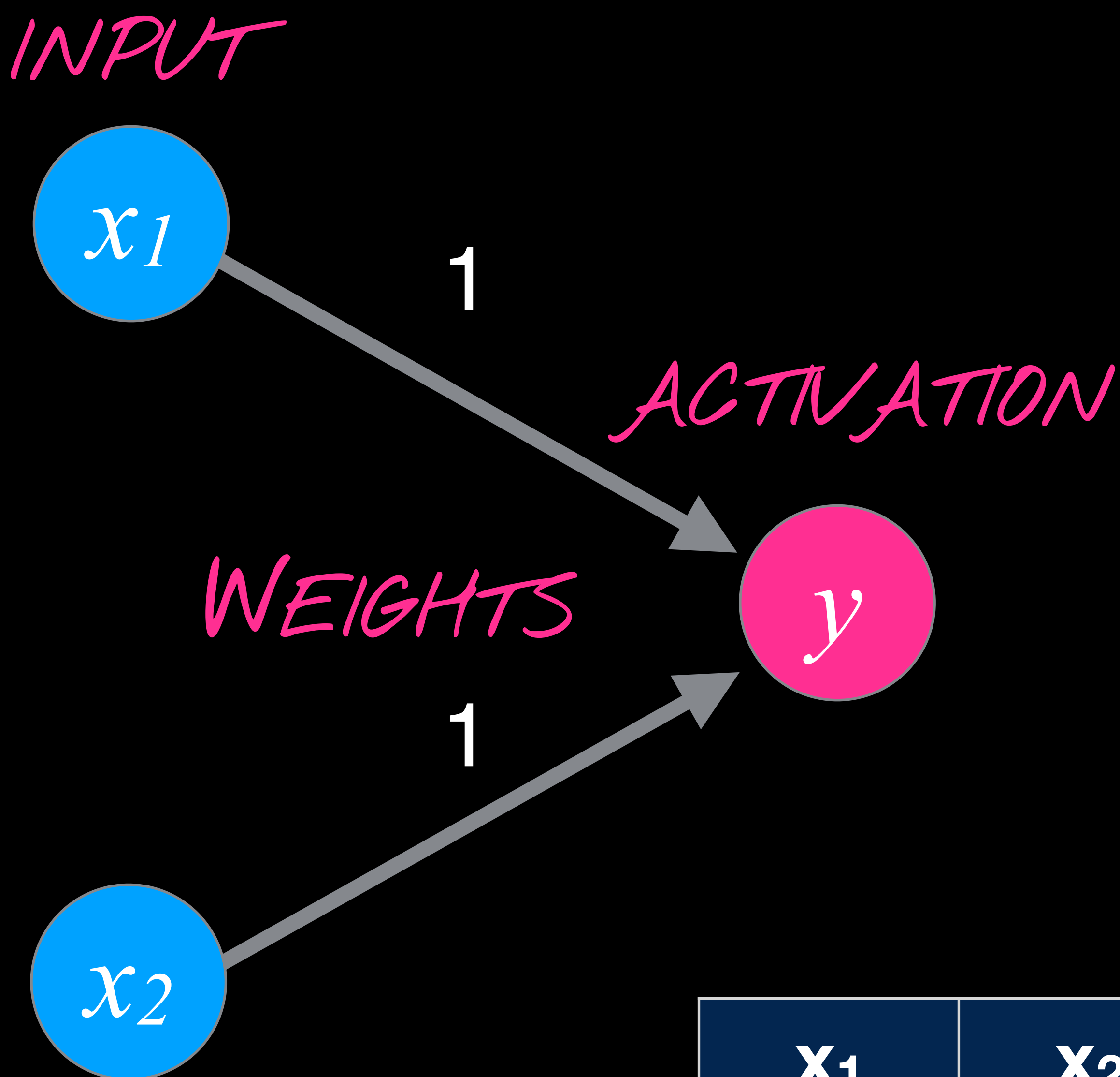
The Perceptron fails at learning even basic concepts

The Perceptron

A Threshold Unit



AND-Perceptron



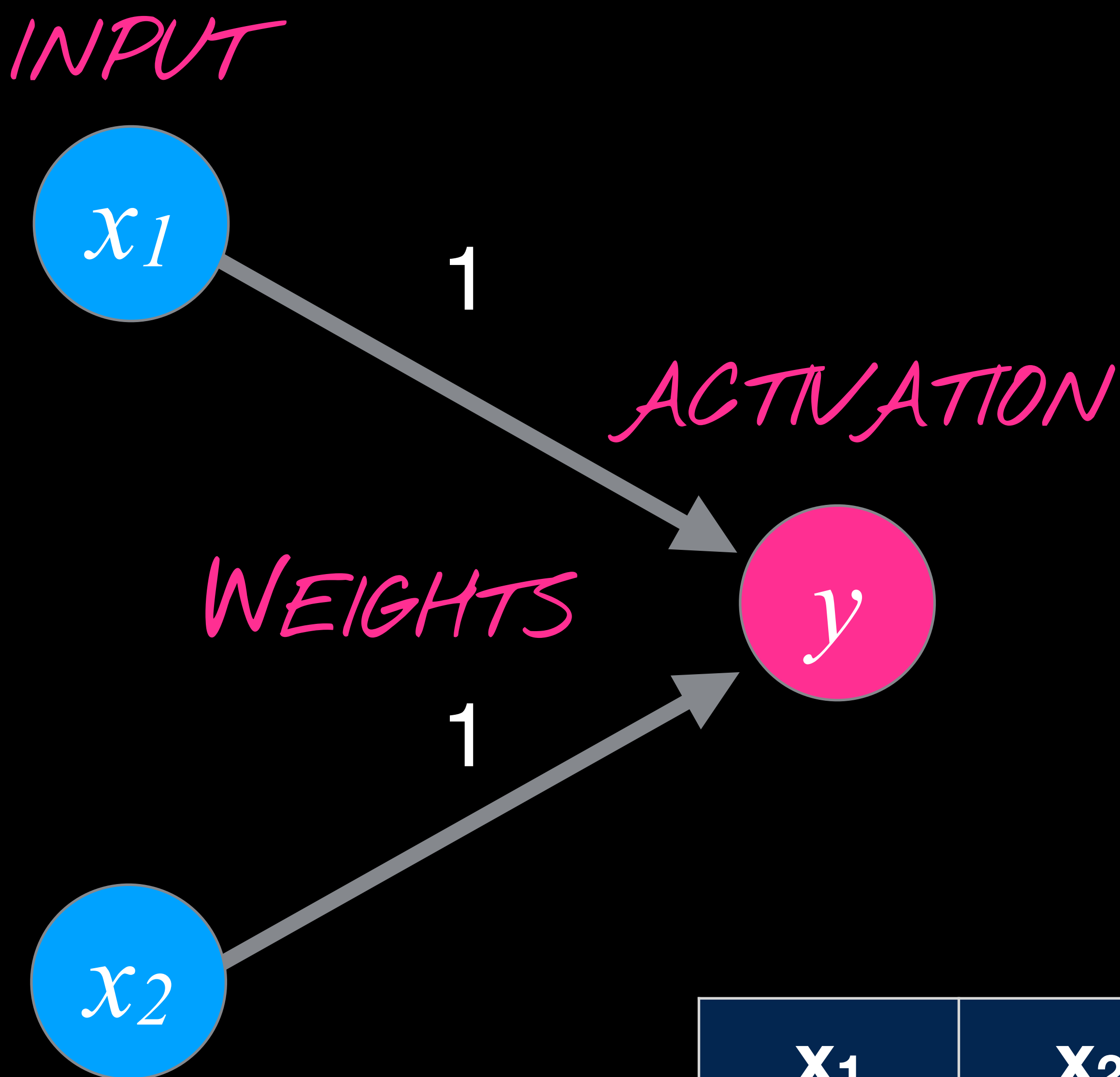
$$f(X) = w_1 x_1 + w_1 x_2$$

$$\hat{y} = \begin{cases} +1 & \text{if } f(X) \geq 2 \\ -1 & \text{otherwise} \end{cases}$$

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

Bocconi

OR-Perceptron

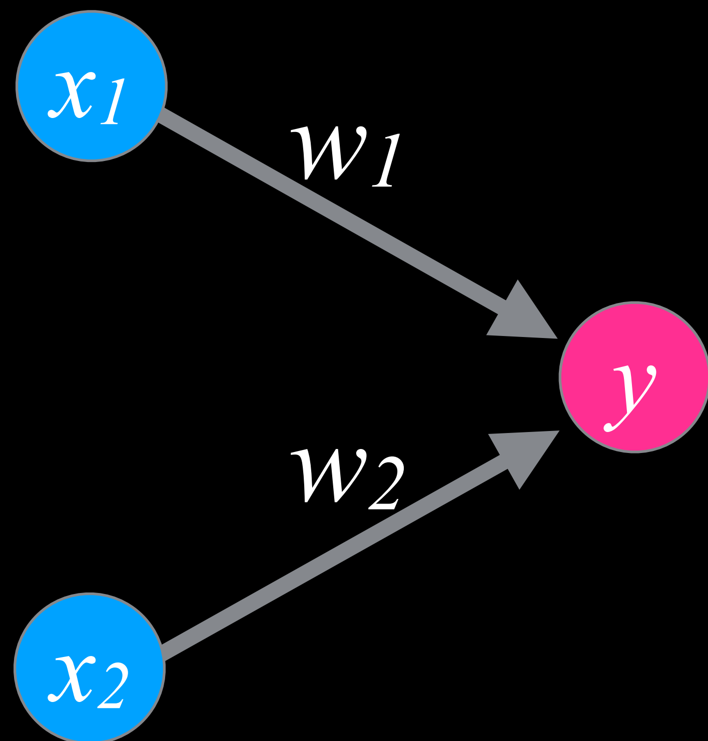


$$f(X) = w_1 x_1 + w_1 x_2$$

$$\hat{y} = \begin{cases} +1 & \text{if } f(X) \geq 1 \\ -1 & \text{otherwise} \end{cases}$$

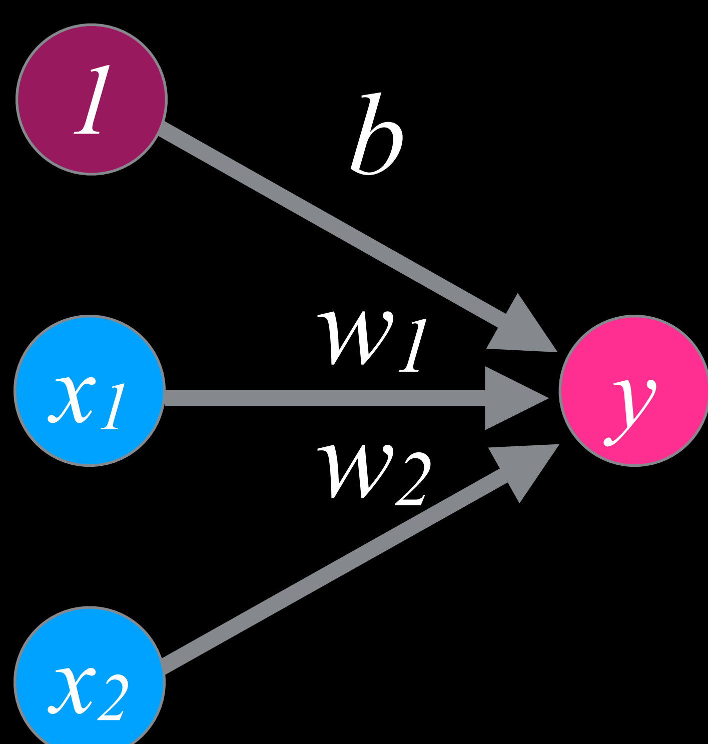
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

Learn the Threshold



$$f(X) = w_1 x_1 + w_1 x_2$$

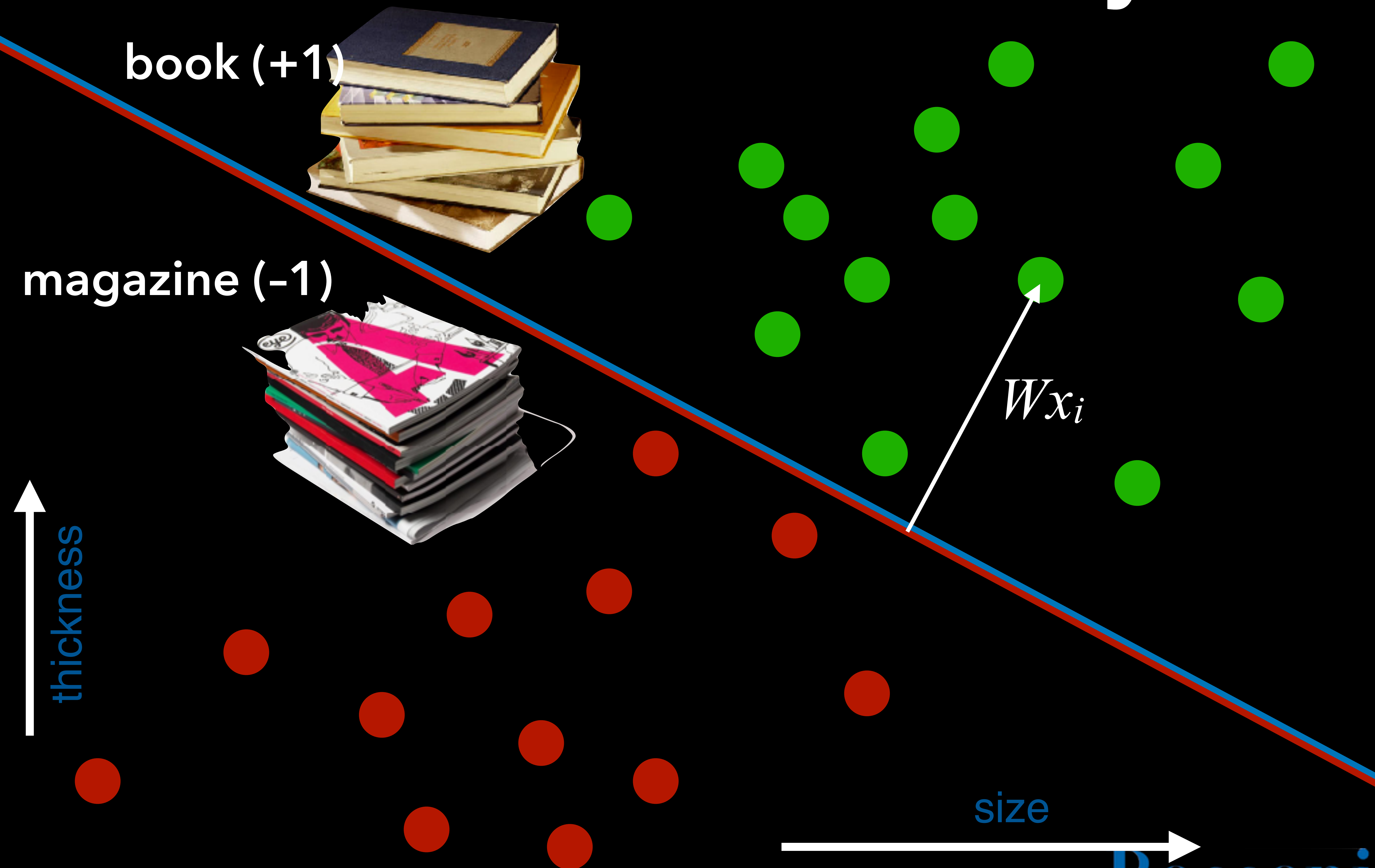
$$\hat{y} = \begin{cases} +1 & \text{if } y(X) \geq t \\ -1 & \text{otherwise} \end{cases}$$



$$f(X) = w_1 x_1 + w_1 x_2 + \mathbf{b}$$

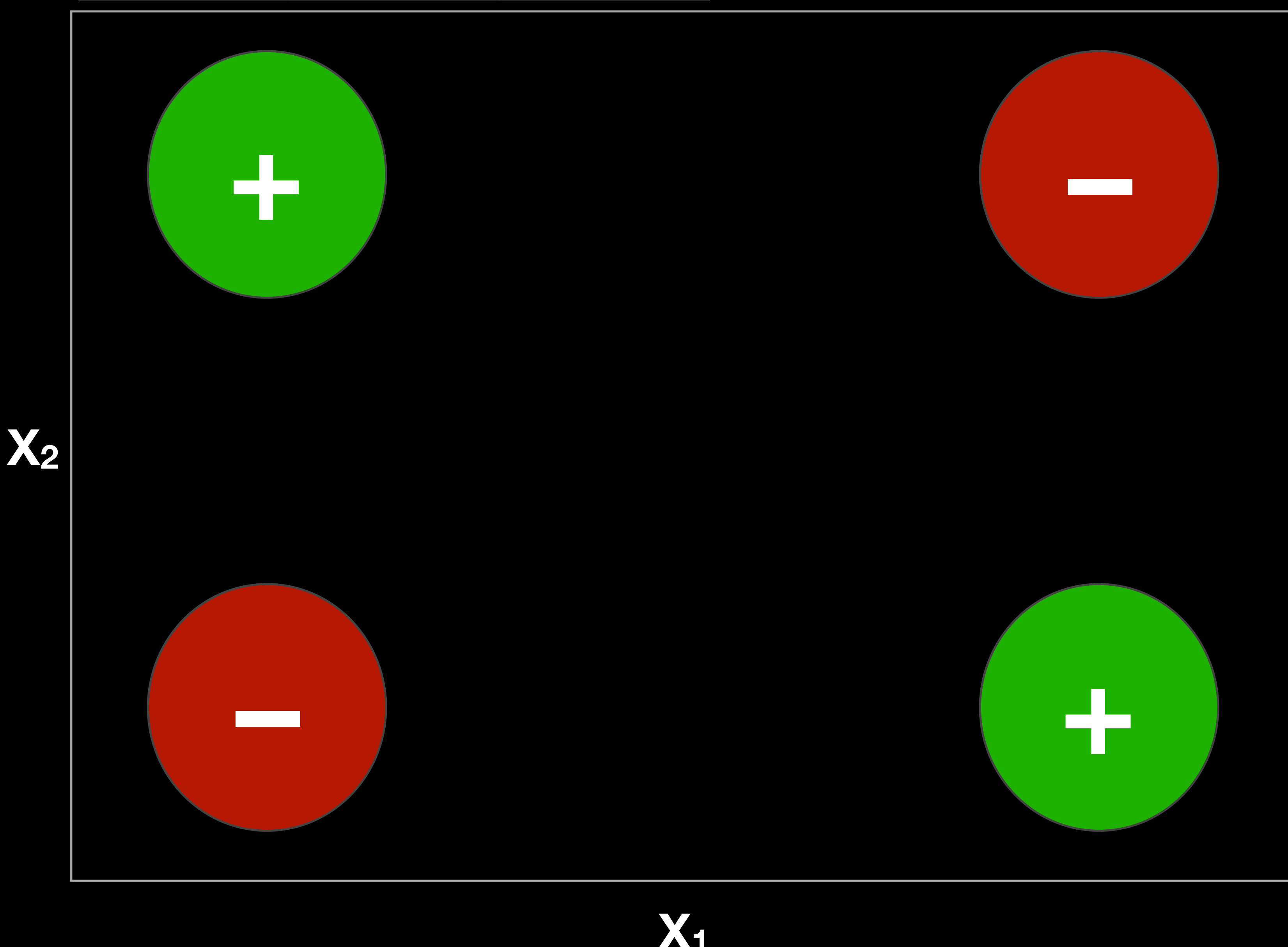
$$\hat{y} = \begin{cases} +1 & \text{if } y(X) \geq \mathbf{0} \\ -1 & \text{otherwise} \end{cases}$$

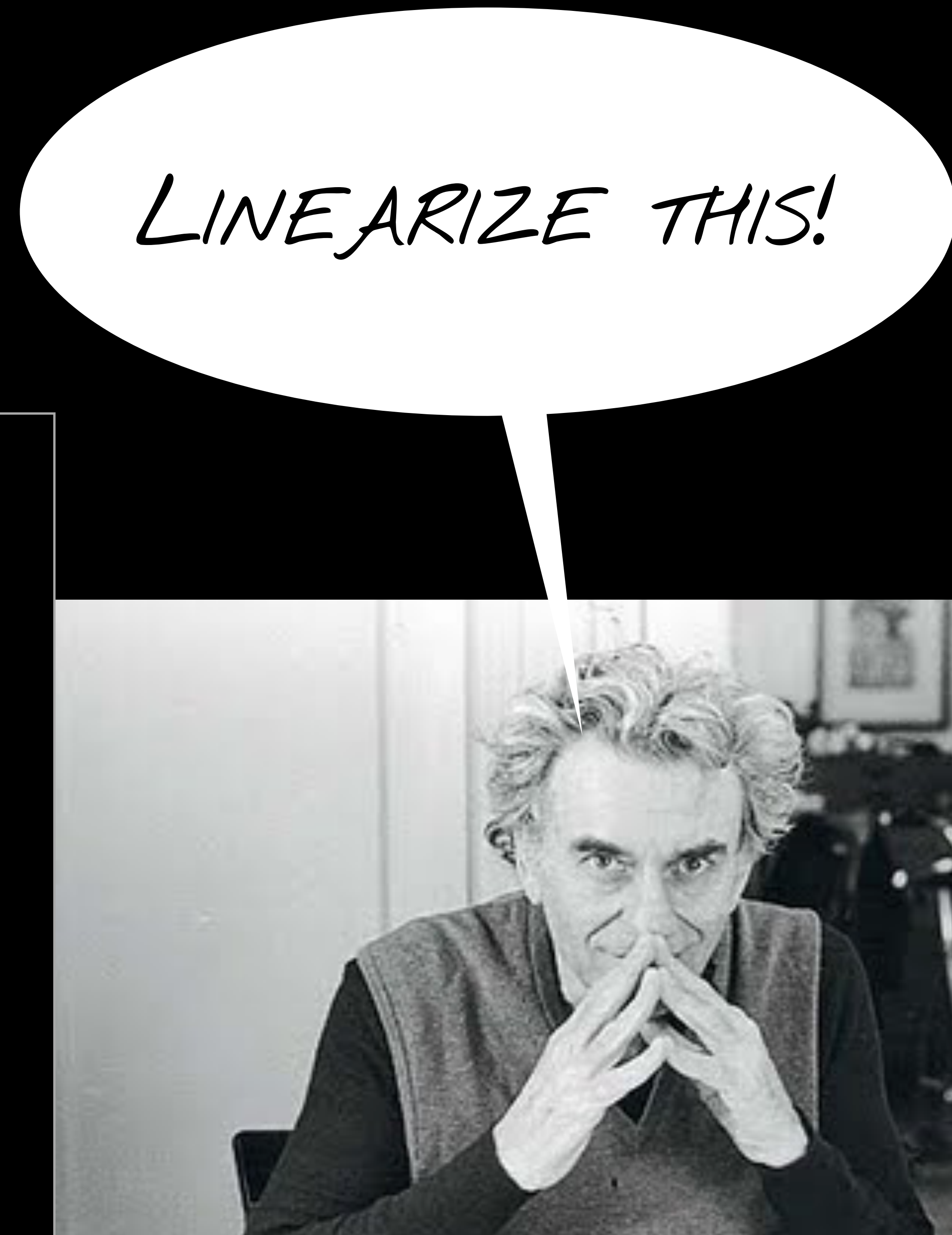
Decision Boundary



The XOR Limit

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

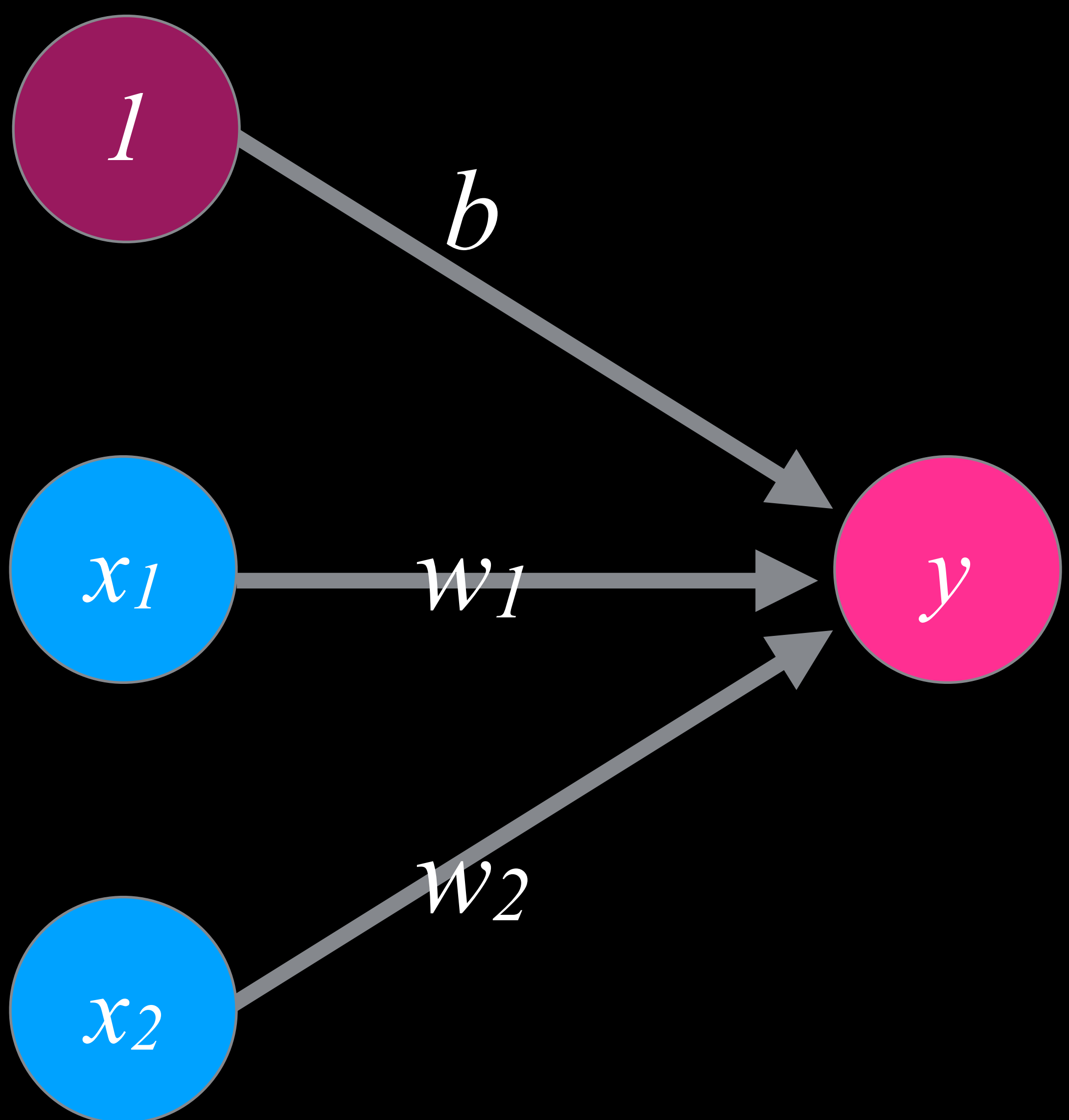




Marvin Minsky
(1927–2016)

Step 1:
Non-Linearity

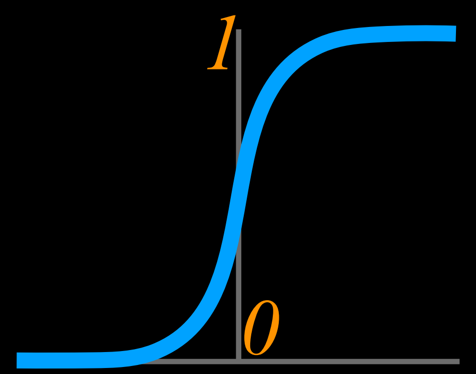
Nonlinear Activation Functions



$$f(X) = \boldsymbol{a}(w_1 x_1 + w_2 x_2 + b)$$

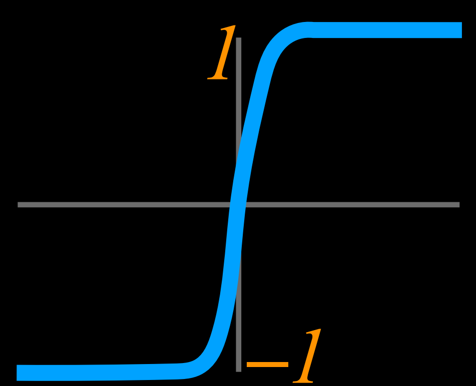
Sigmoid

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



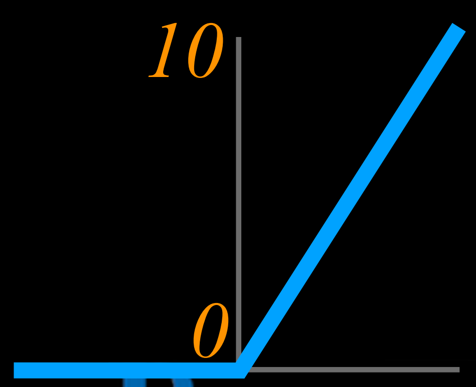
tanh

$$tanh(x)$$



ReLU

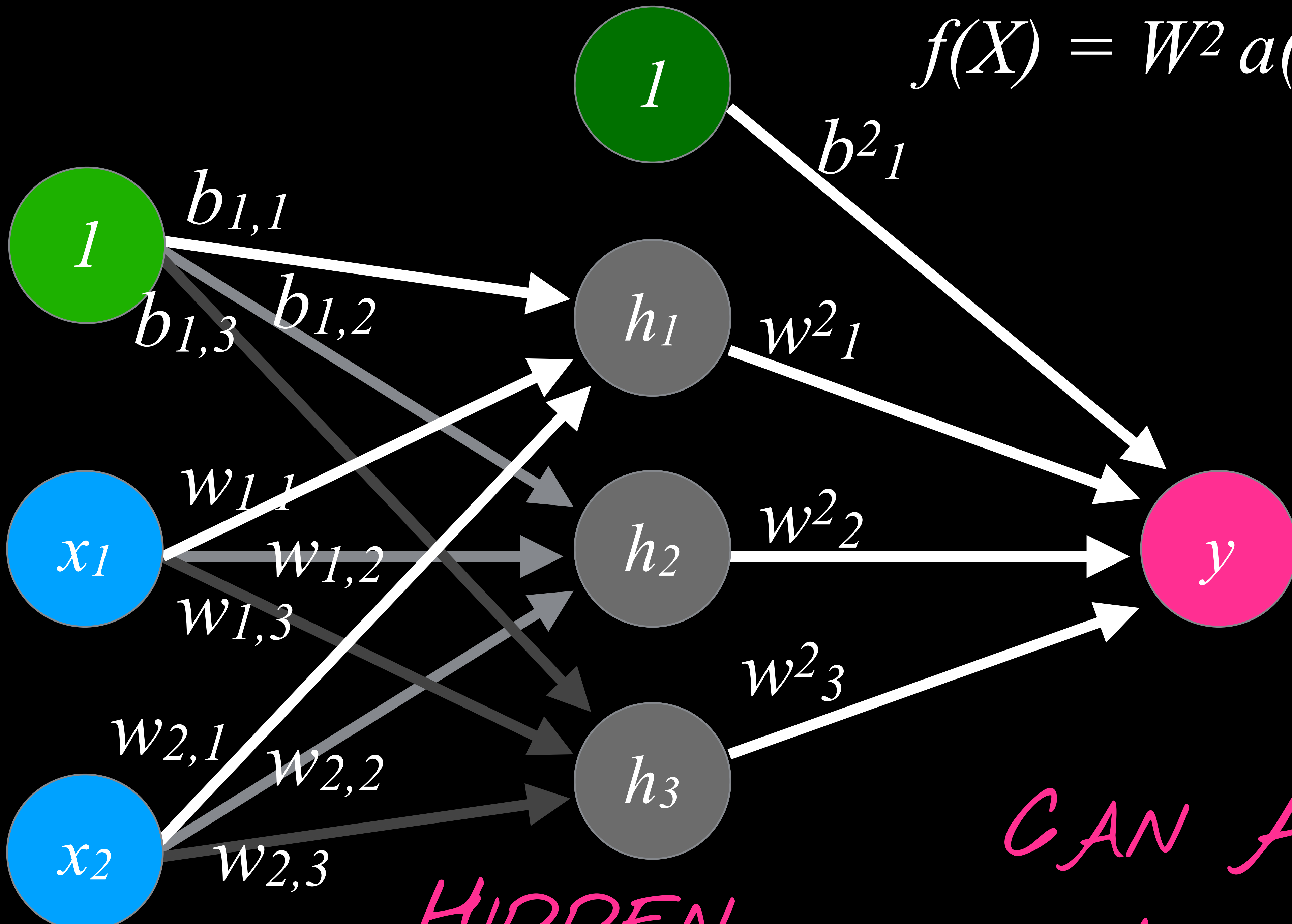
$$max(0, x)$$



Step 2:
Going Deep –
The Multilayer Perceptron

Multilayer Perceptron

$$f(X) = W^2\, a(W^1 X + b^1) + b^2$$



HIDDEN LAYER

- How many perceptrons do you see?
- How many parameters?

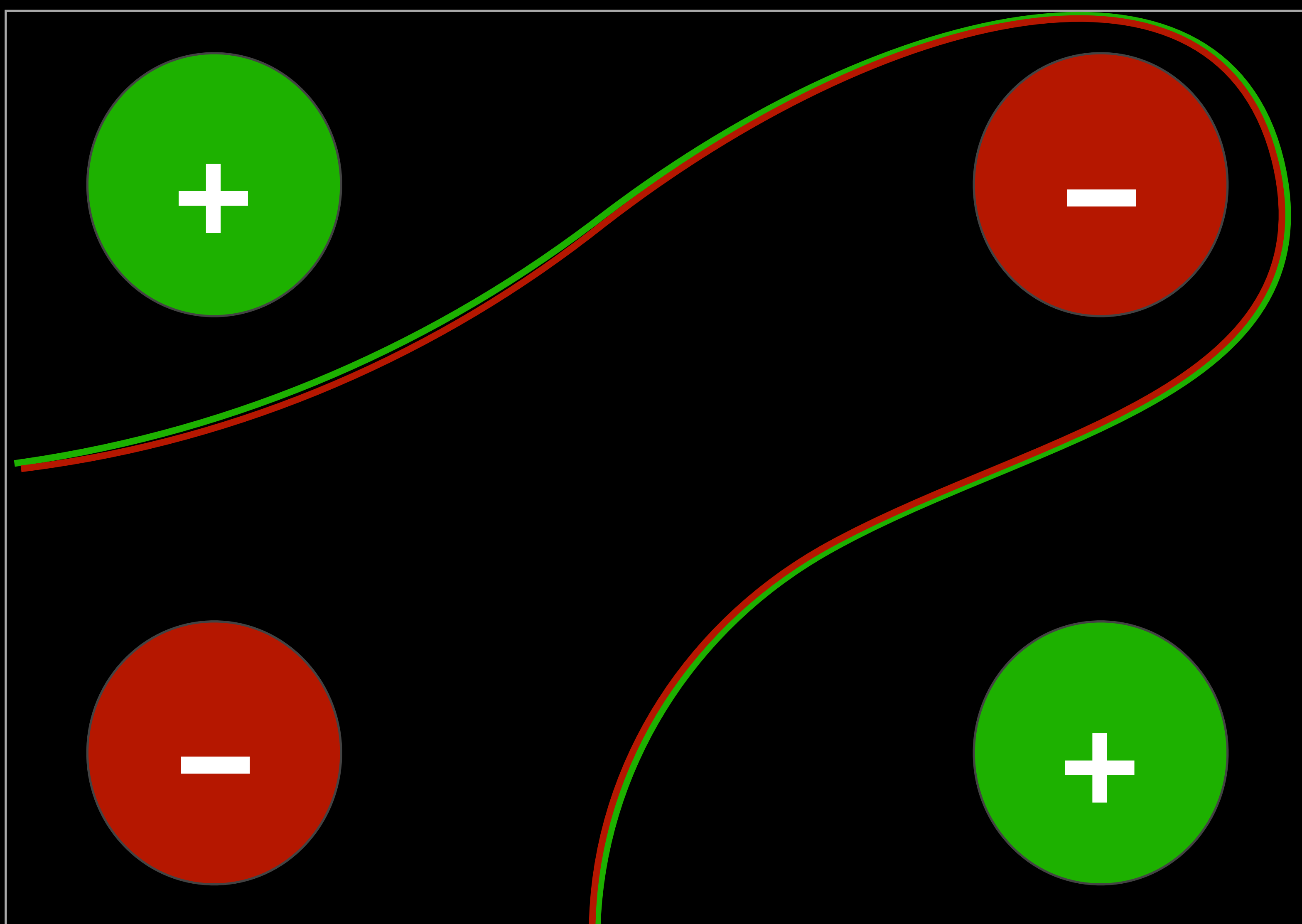
CAN APPROXIMATE ANY FUNCTION!

The XOR Limit

x_1	x_2	
0		
1		
0	1	
1	1	0

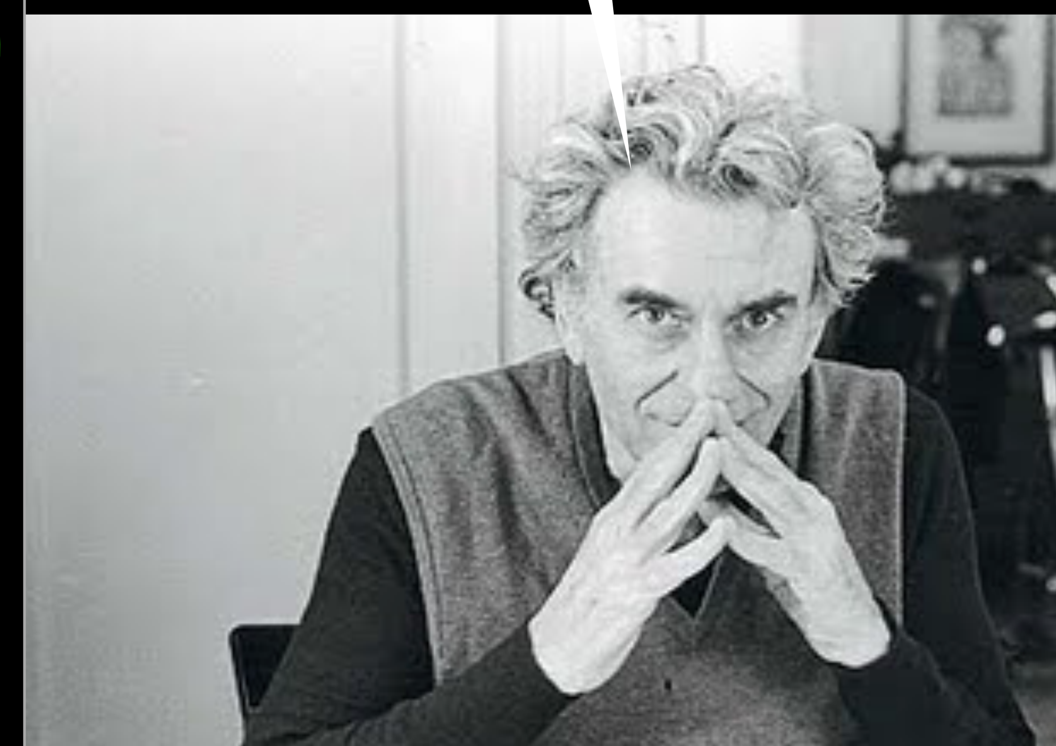
THERE YOU GO...

LINEARIZE THIS!



x_2

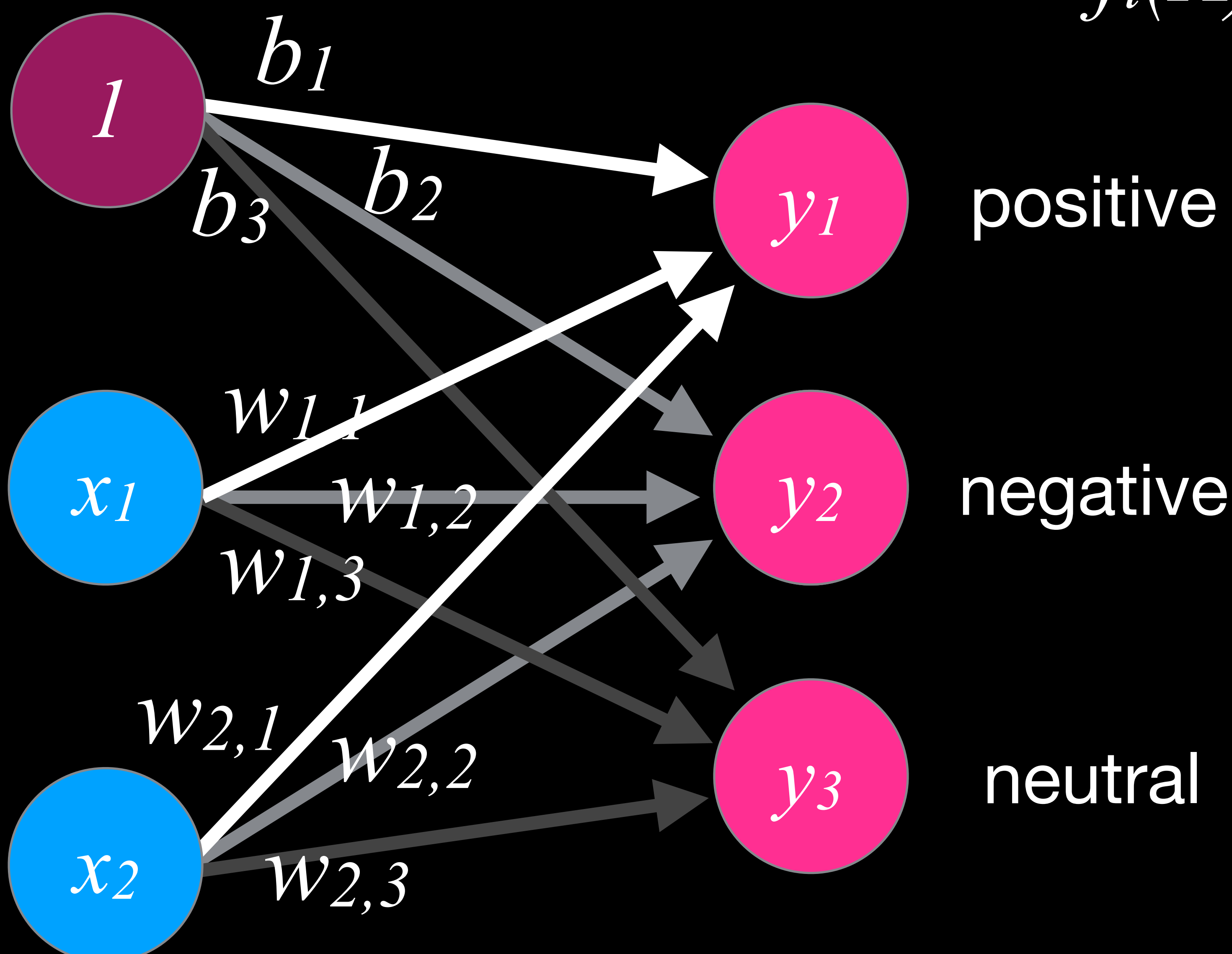
x_1



Marvin Minsky (1927–2016)

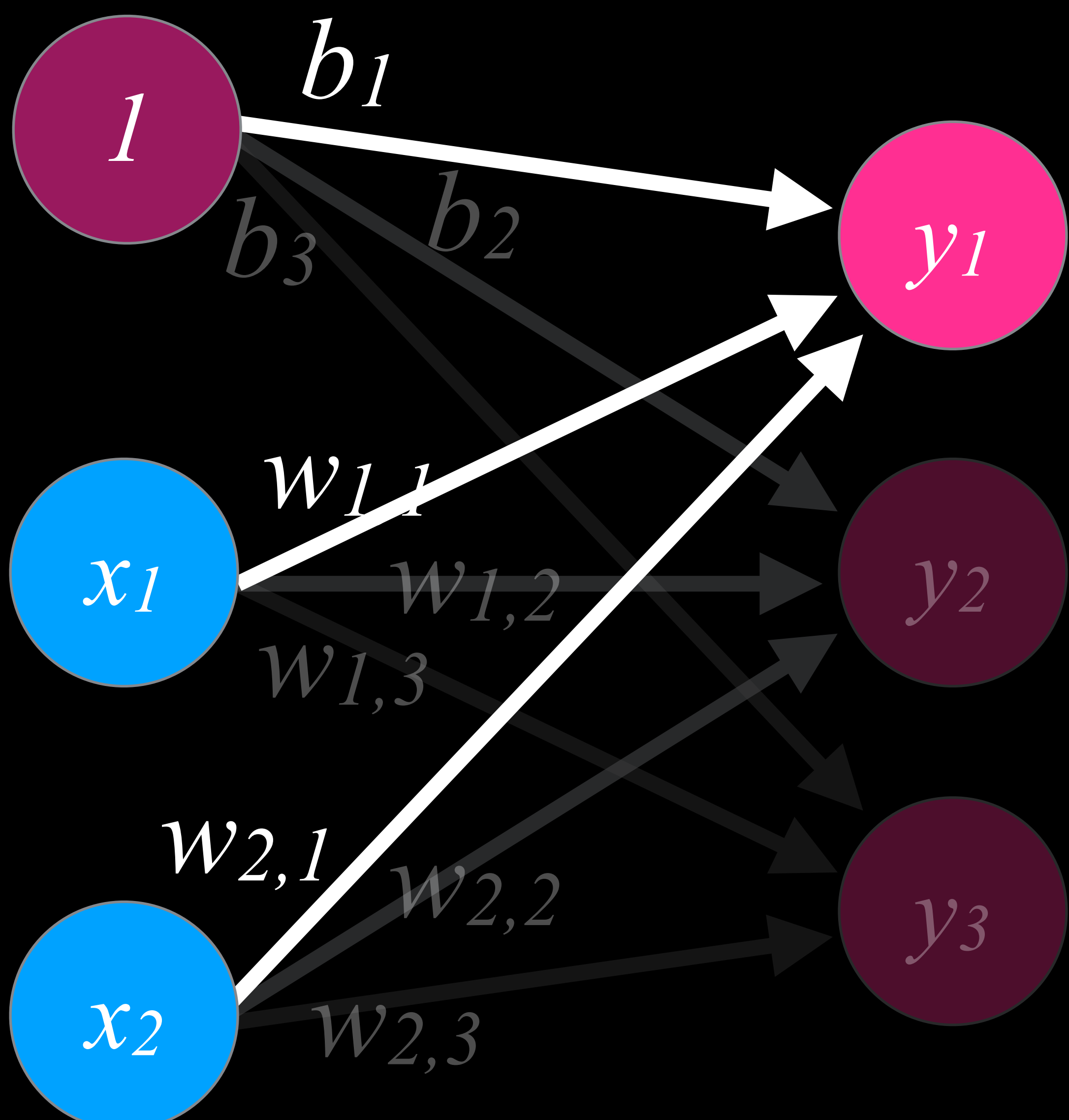
Multi-Class Output

$$f_i(X) = a(w_{1,i}x_1 + w_{2,i}x_2 + b_i)$$



$$\hat{y} = \underset{i}{argmax}\, f_i(X)$$

Enter the Matrix



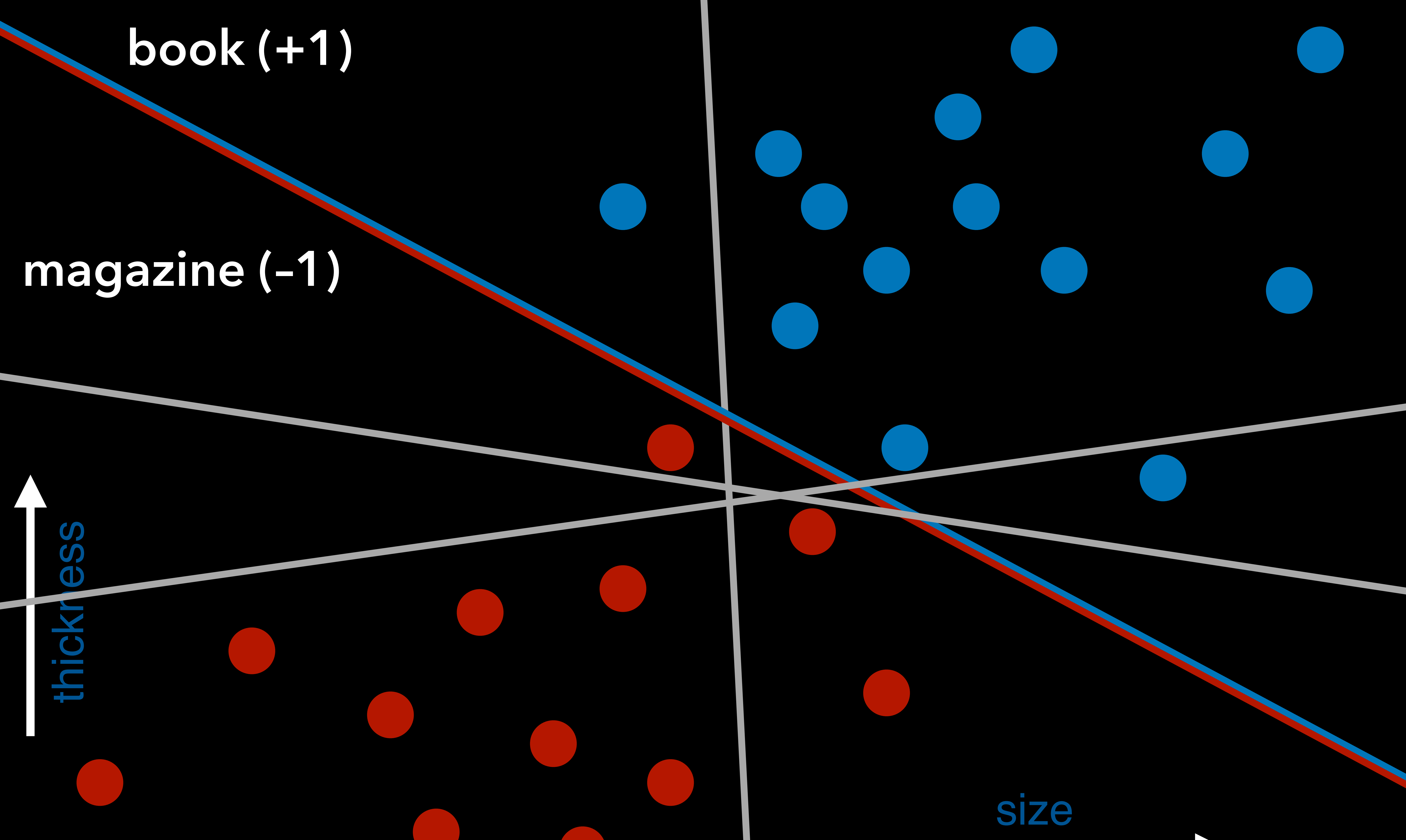
$$y_1 = a(w_{1,1}x_1 + w_{2,1}x_2 + b_1)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$Y = a(W^i X + b^i)$$

Learning

Decision Boundary



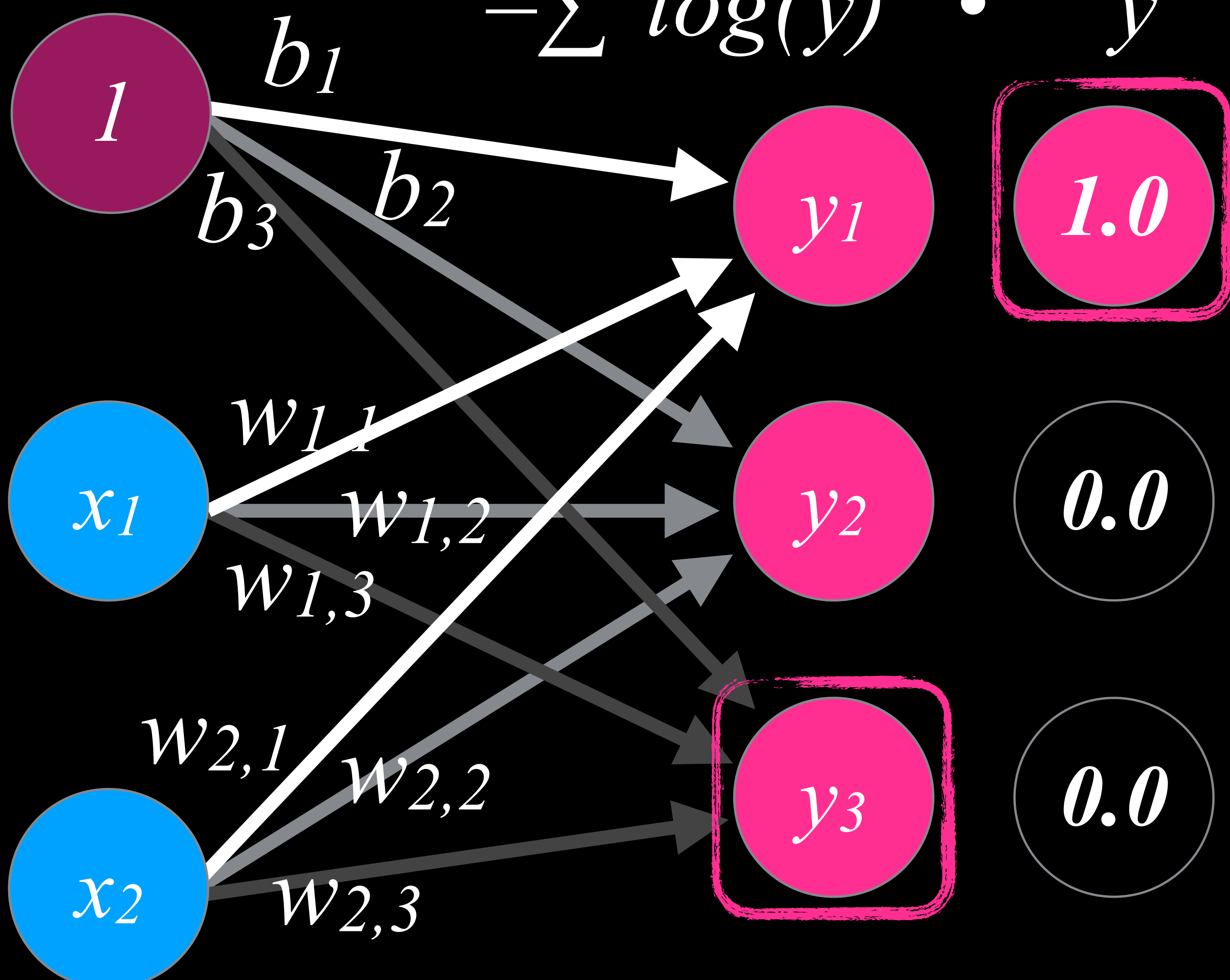
Bocconi

Error!

neutral positive

$$-\sum log(\hat{y}) \bullet y$$

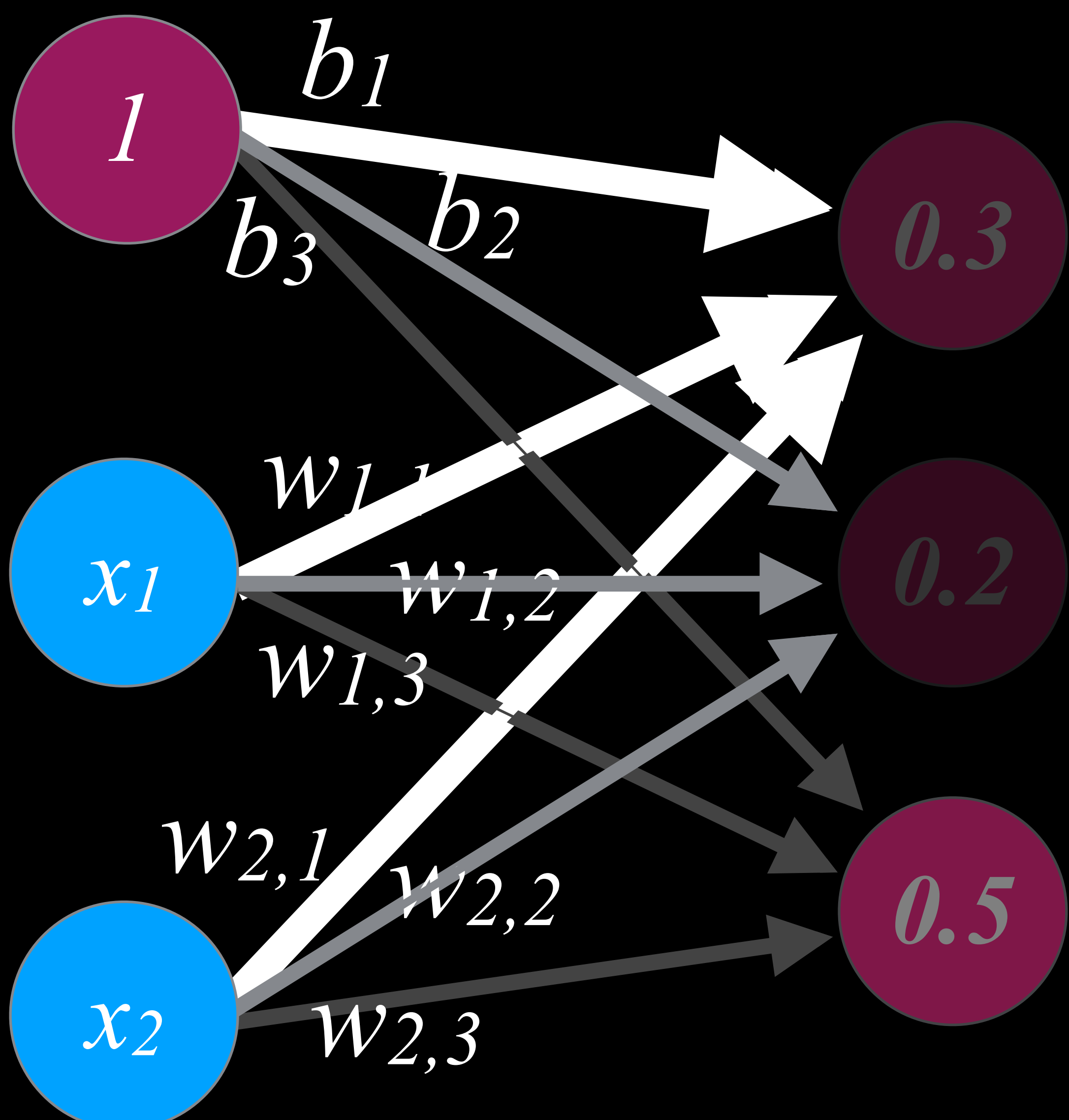
HOW FAR OFF ARE WE?



$$-1.2 + 0.0 + 0.0 = 1.2$$

CROSS-ENTROPY

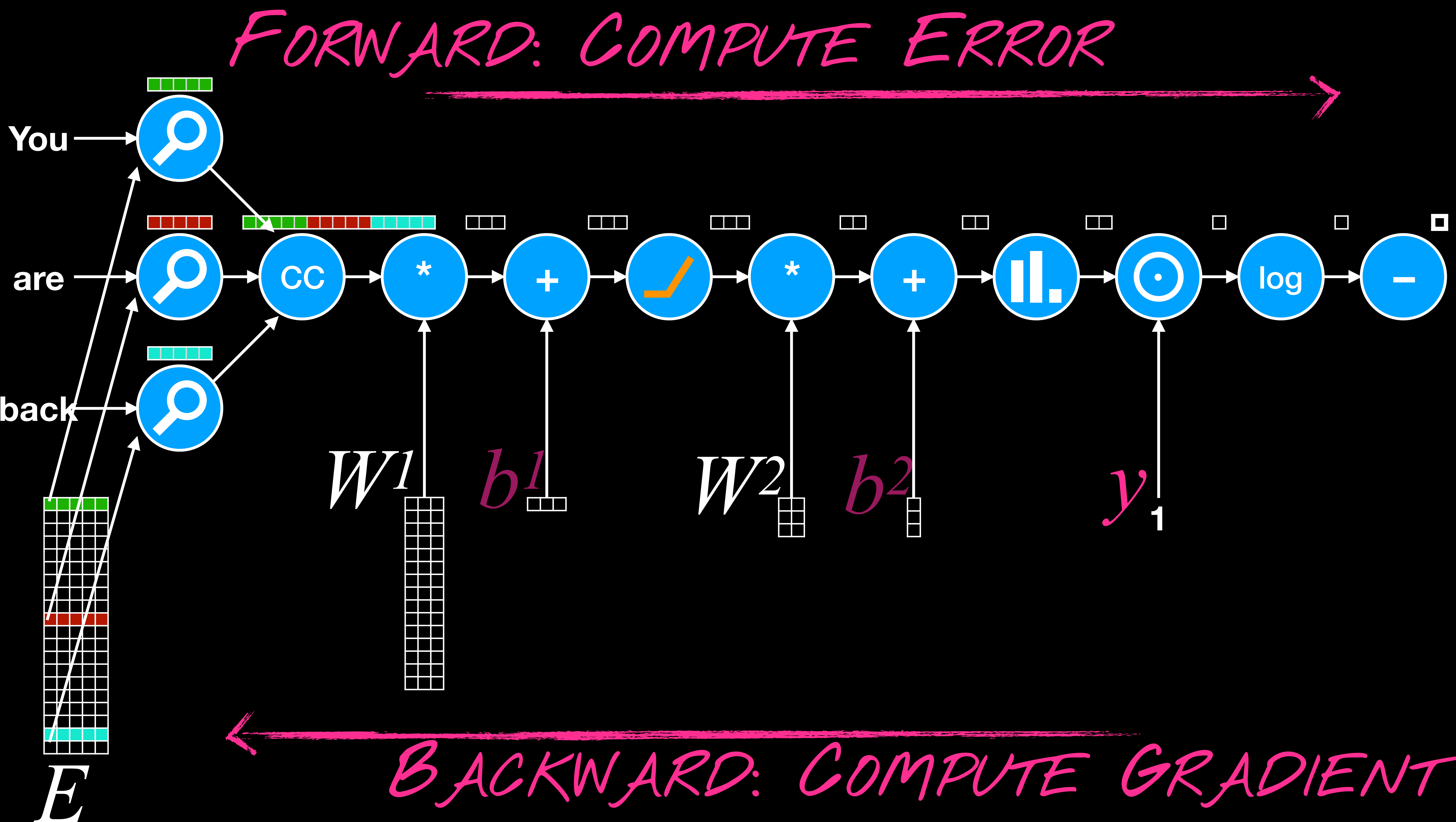
Backpropagation



1.2

- Adjust weights/bias proportionately to change, using **Stochastic Gradient Descent**
- In deeper networks: compute effect on previous layer activation, then adjust *their* incoming weights accordingly, using **Chain Rule**
- If input layer is adjusted as well = learning representations

Computational Graph



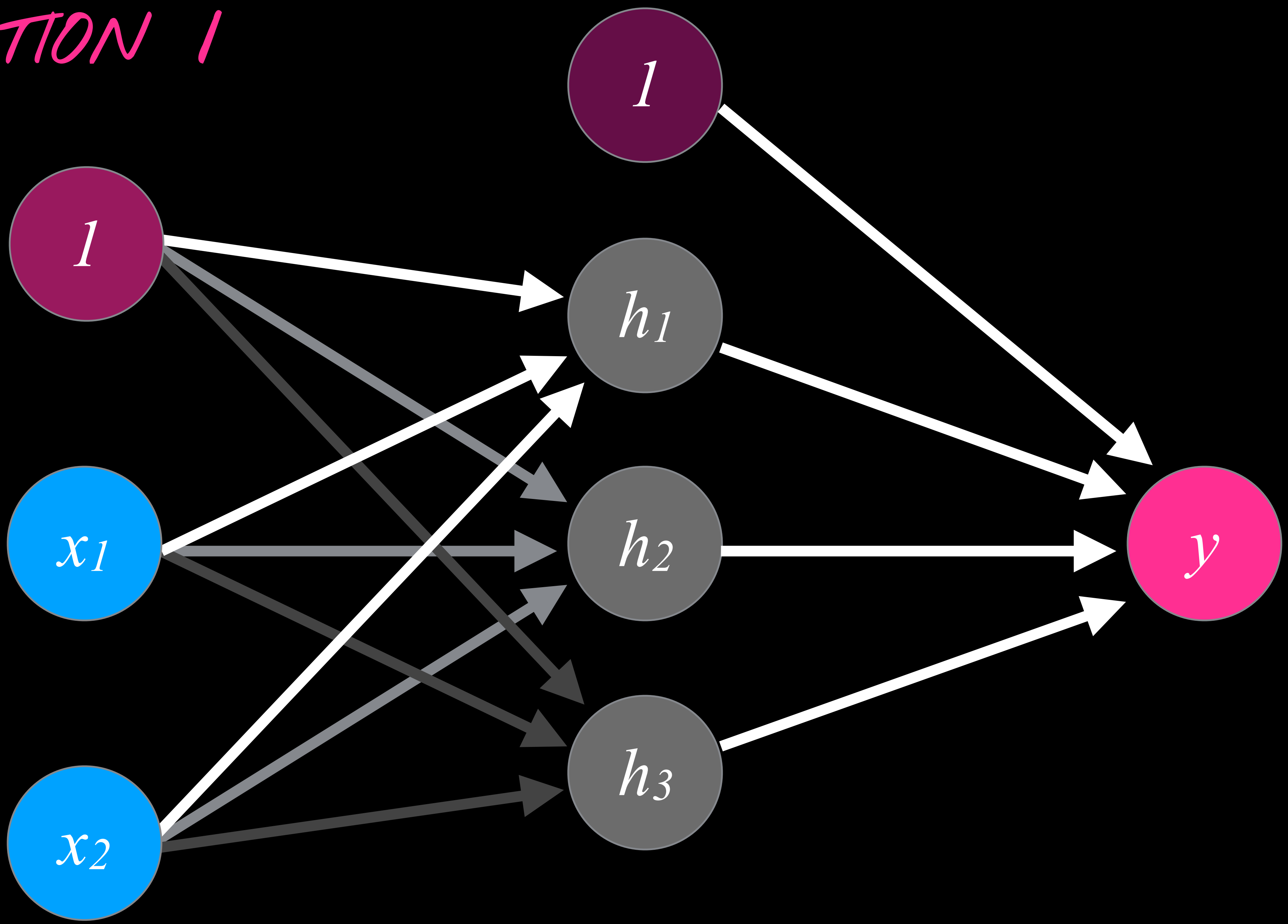
Regularization with Dropout

Overfitting

- ALVINN autonomous vehicle, trained on a particular stretch of road
- Drove off the road when turned around => focused on having a ditch on one side
- Idea: randomly remove nodes to avoid over-reliance

Dropout

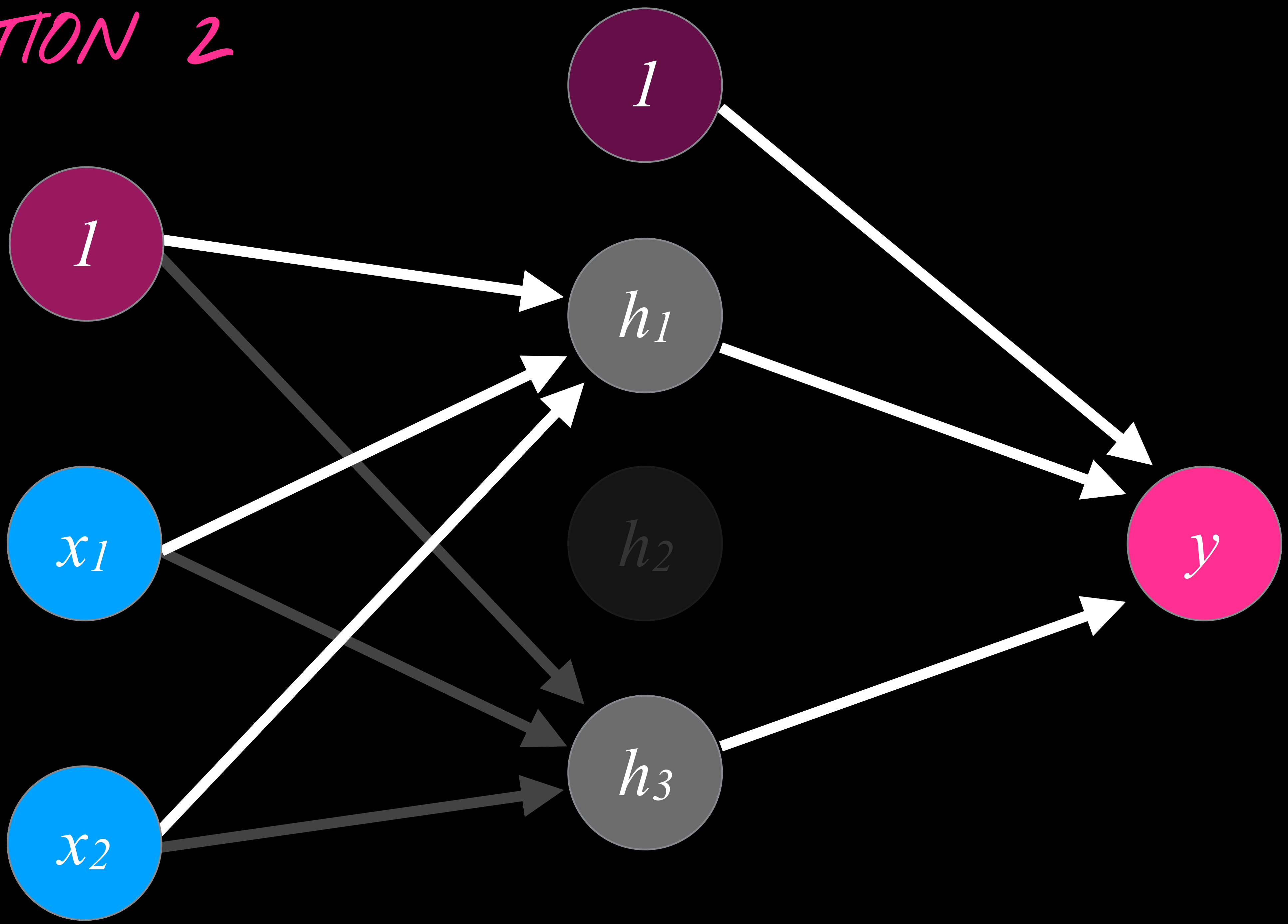
ITERATION 1



Bocconi

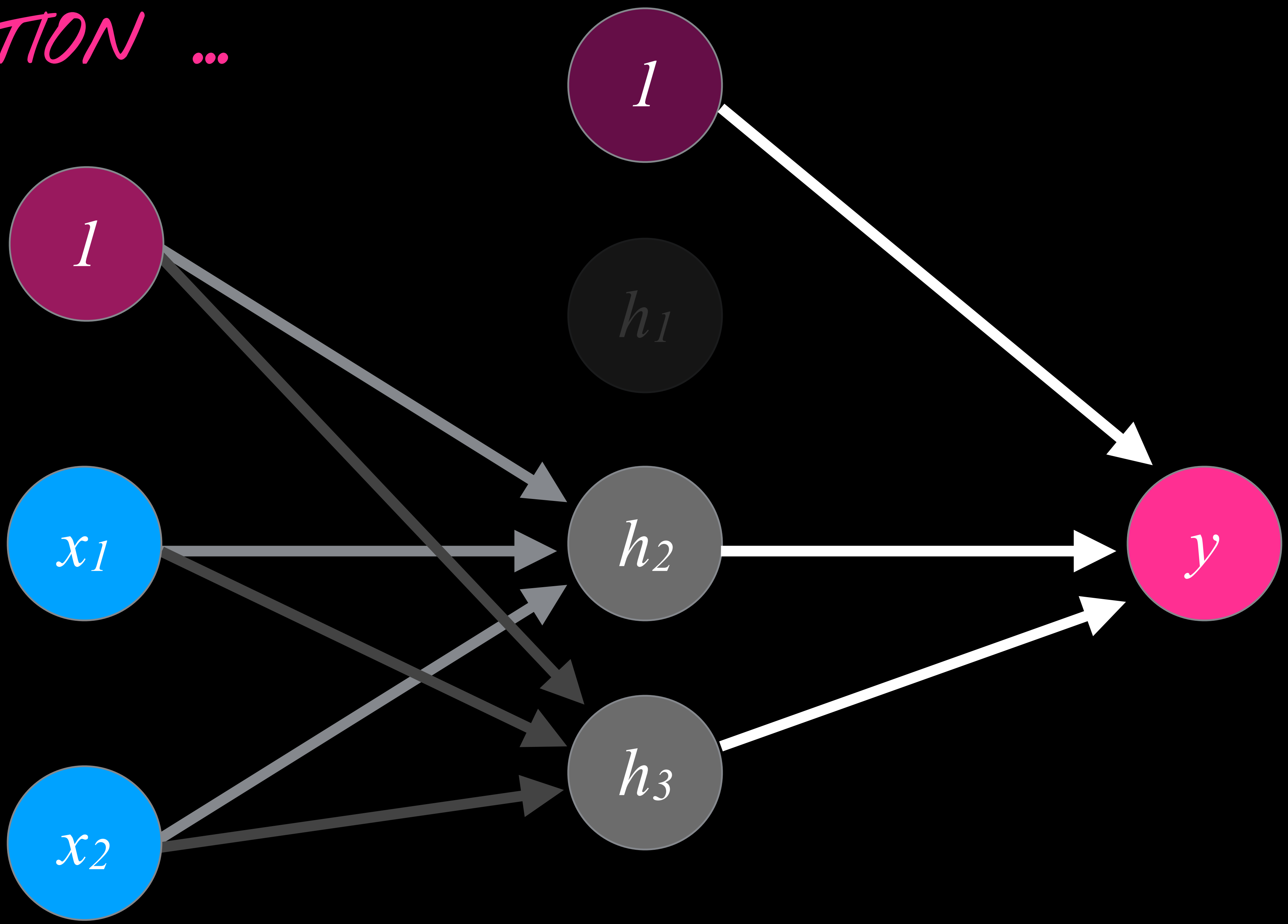
Dropout

ITERATION 2



Dropout

ITERATION ...



Bocconi

Wrapping up

Take Home Points

- The **perceptron** is the basic building block of NNs
- Several perceptrons are a **Multilayer Perceptron** or **Feedforward Network**
- Each layer is matrix multiplication wrapped in an **activation function** (usually **ReLU**)
- Training **backpropagates** an error through the network to change weights
- **Dropout** helps regularize networks by randomly deleting nodes

Bocconi

Moar Sources

- 3Blue1Brown: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

- Yoav Goldberg Primer: https://arxiv.org/pdf/1510.00726.pdf

- The Keras book

- …

Bocconi