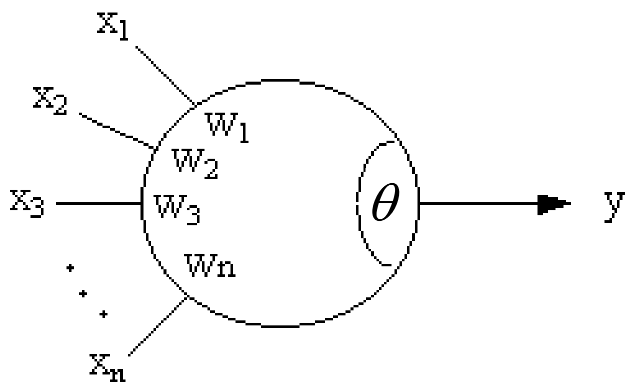
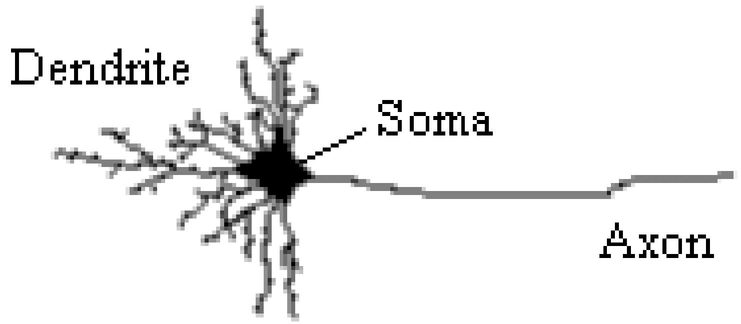
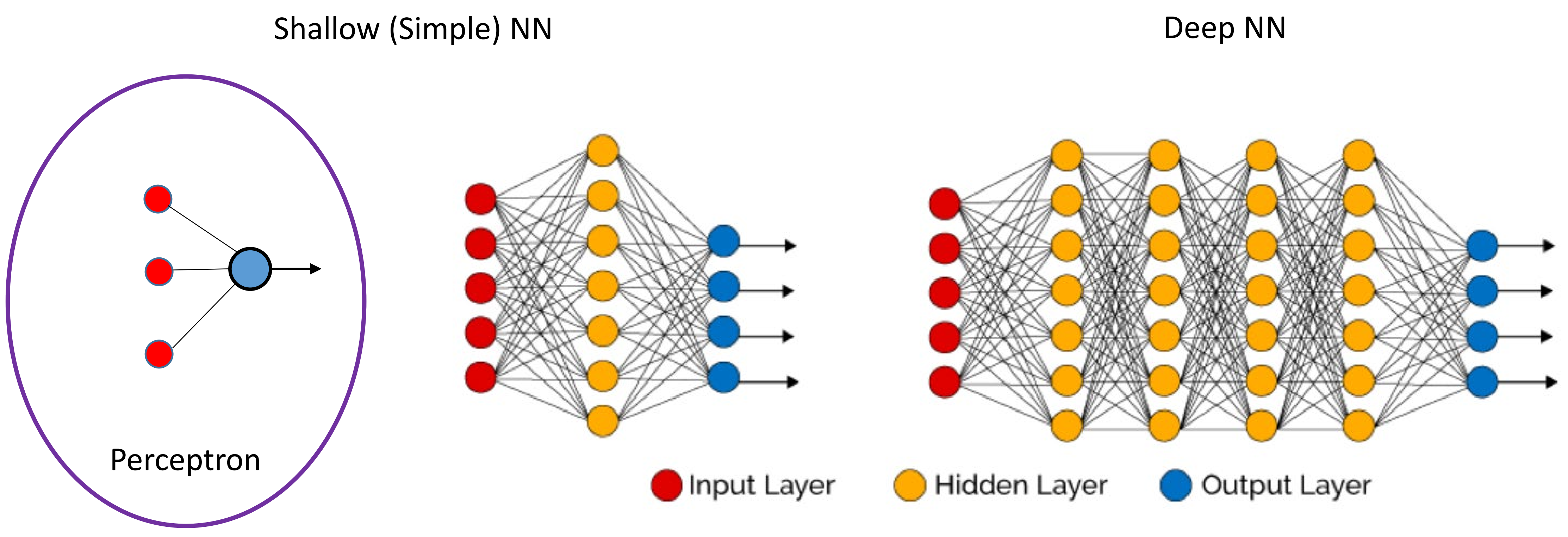
Perceptron – One Artificial Neuron





- Architecture
- Function
- *How to train a perceptron*

Brain-Inspired Neural Networks, Overview



↑ This lecture

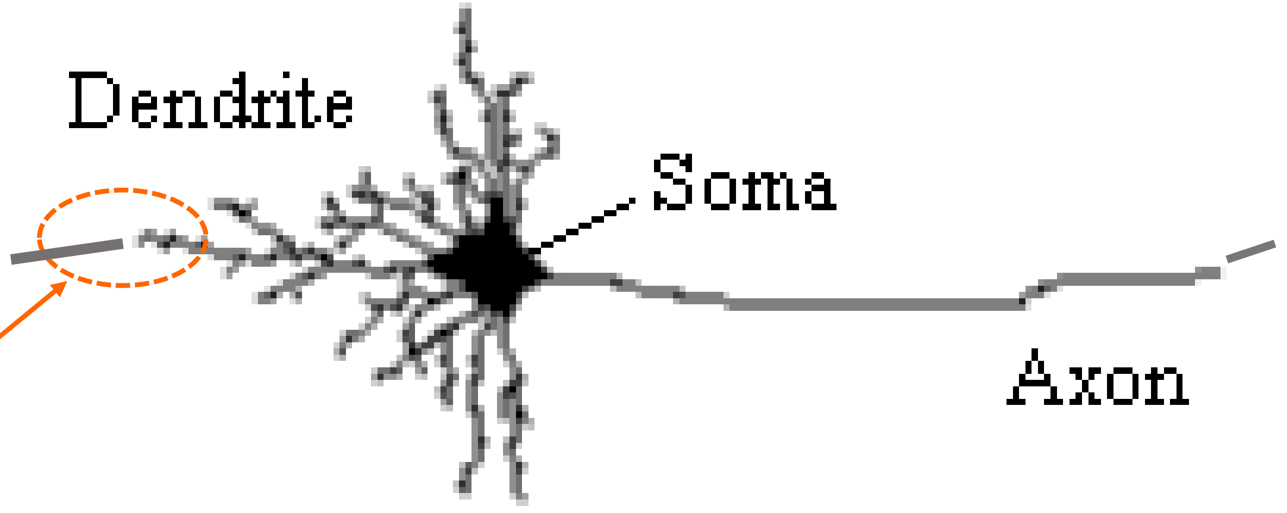
ANN (Artificial Neural Networks)

- Artificial Neural Network is a system loosely modeled on the human brain
- This is a field to attempt to simulate multiple layers of simple processing unit called neurons
- Each neuron is linked to certain of its neighbors with varying coefficients of connectivity that represent the strength of the connections
- Learning is accomplished by adjusting these strengths to cause overall network to output appropriate results

History

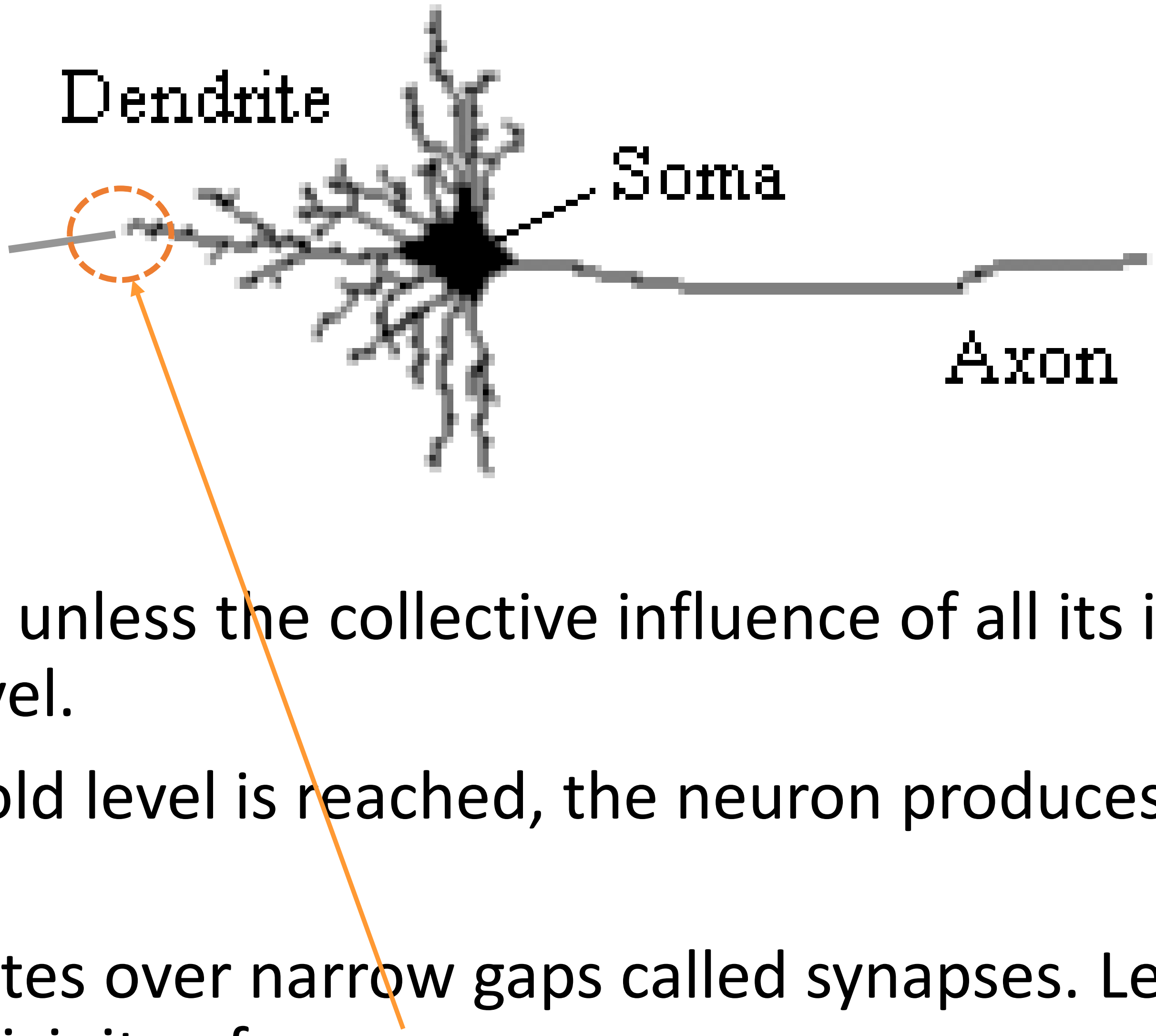
- 1890 – W. James & Psychologist, both thoughts and body activity resulted from interactions among neurons within the brain
- 1936 – A. Turing, brain as computing paradigm
- 1943 – McCulloch & Pitts, Artificial Neuron, concept of perceptron
- 1949 – Hebb, Hebbian Learning Rule
- 1957 – Rosenblatt, First hardware implementation of Perceptron
- 1969 – Minsky & Papert, Perceptron Critique
- 197x – Few studied about NN
- 1982 – Hopfield Network
- 199x – ANN with Evolutionary Algorithms
- *2015 – TensorFlow (with CNN) released, Nov 9th, a breakthrough*
- *2022 – ChatGPT*

A Neuron



- Dendrite: Accepts input
- Soma: Process the inputs
- Axon: Turn the processed input into output
- Synapse: The **electrochemical** contact between neurons (axon and dendrite)

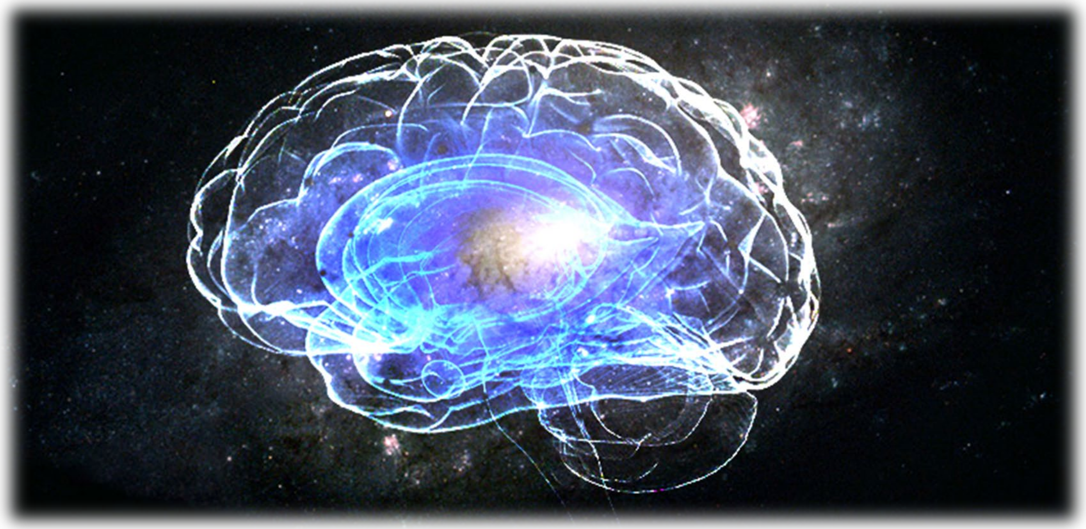
A Neuron

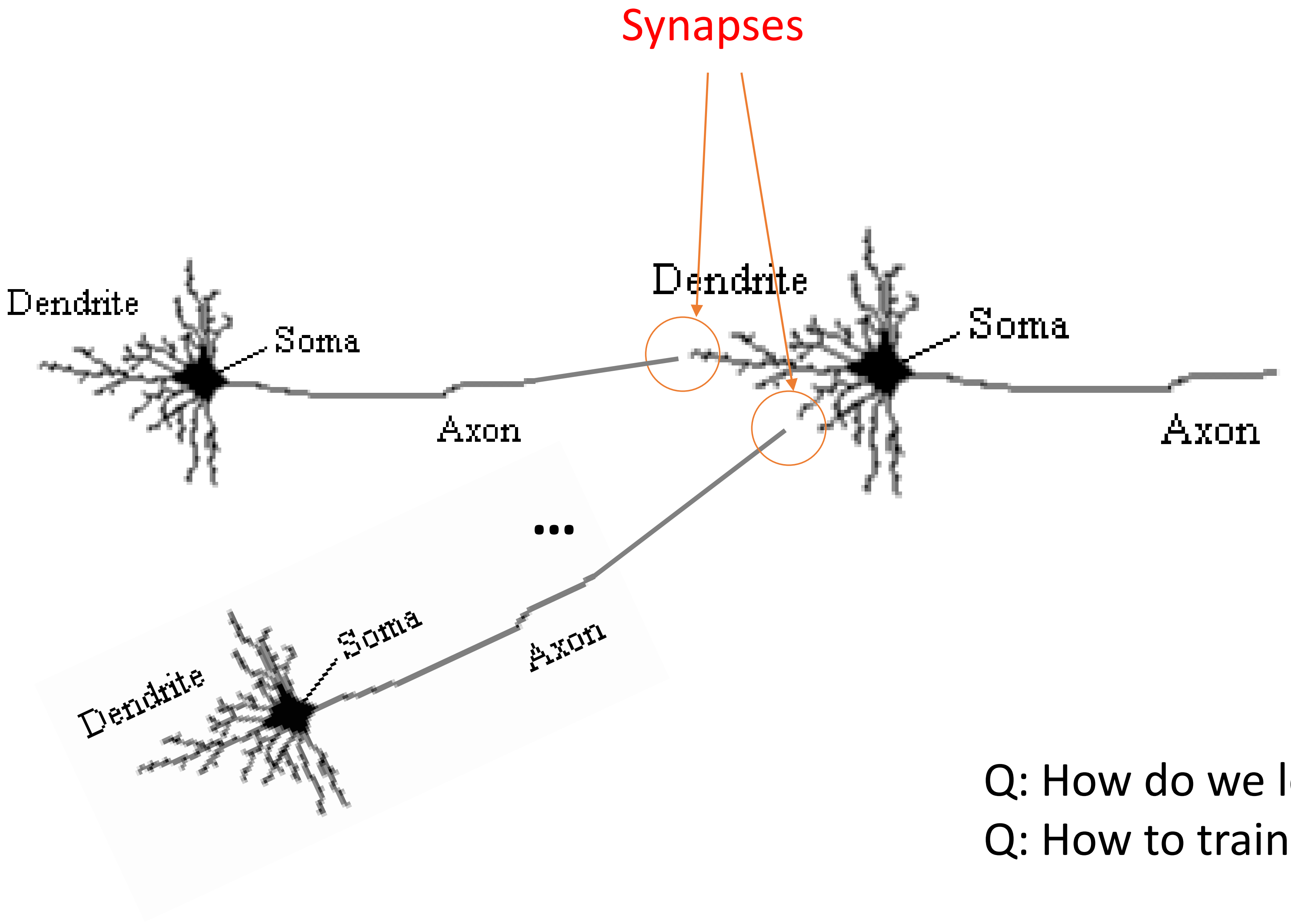


- A neuron does nothing unless the collective influence of all its inputs reaches a threshold level.
- Whenever that threshold level is reached, the neuron produces a full-length output. (fire)
- Axons influence dendrites over narrow gaps called synapses. Learning may take place in the vicinity of synapse.
- About 10^{11} neurons per person.

Why Your Brain is Like The Universe

- Your brain is the most complex, mind-blowing organ in the universe
- It is estimated to have over 100 billion (10^{11}) neurons, which is about the number of stars in the Milky Way Galaxy

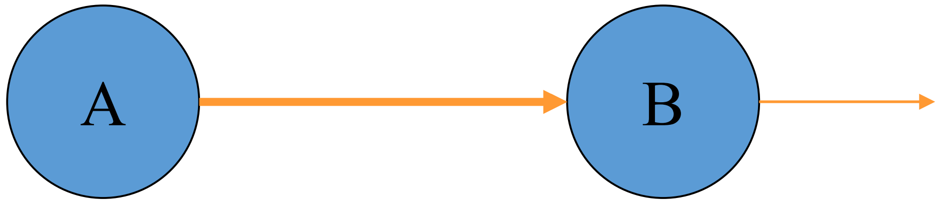




Q: How do we learn?

Q: How to train this network?

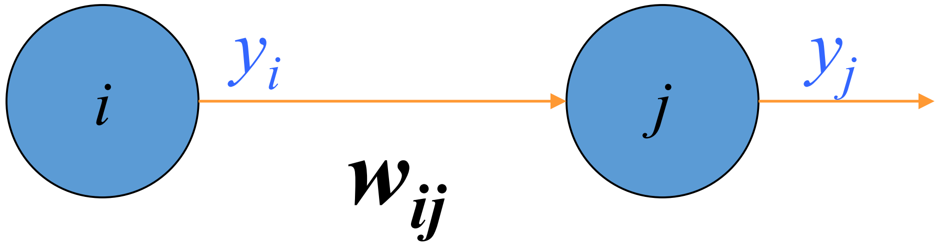
Hebbian Learning Rule



- *When an axon of cell A is near enough to excite a cell B and repeatedly and 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.*
- Organization of Behavior, Hebb, 1949

Hebbian Learning Rule

- In a simple math form Hebbian learning is nothing but correlation computation to adjust weights:

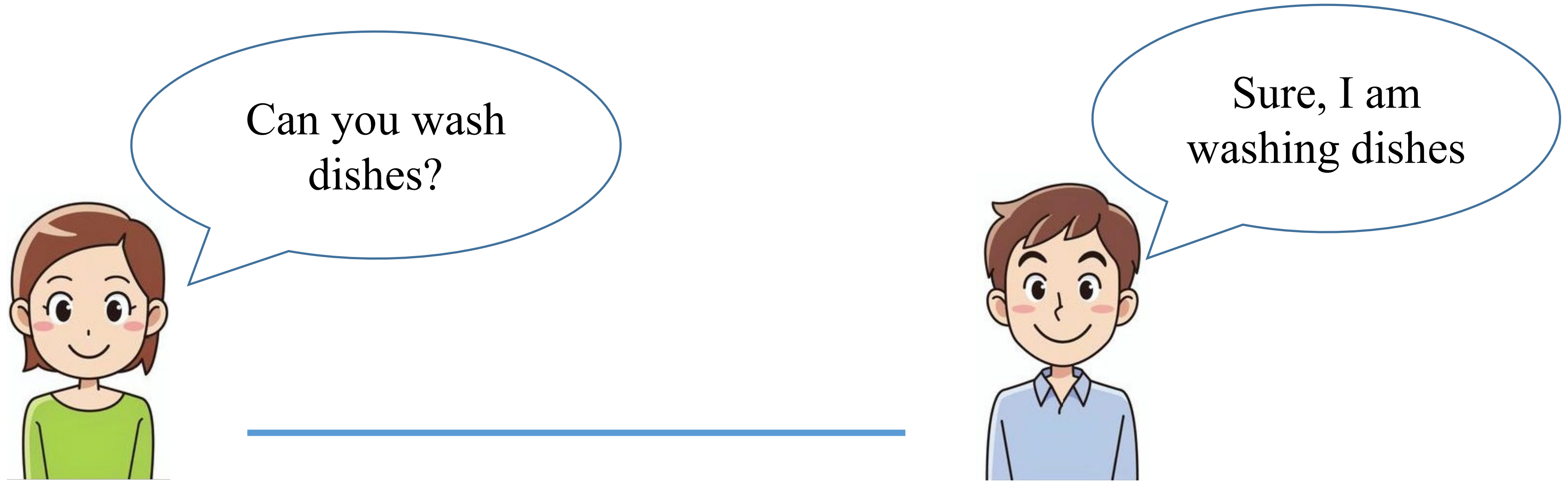


$$\Delta w_{ij} = y_i y_j \quad or$$

$$w_{ij} = w_{ij} + y_i y_j$$

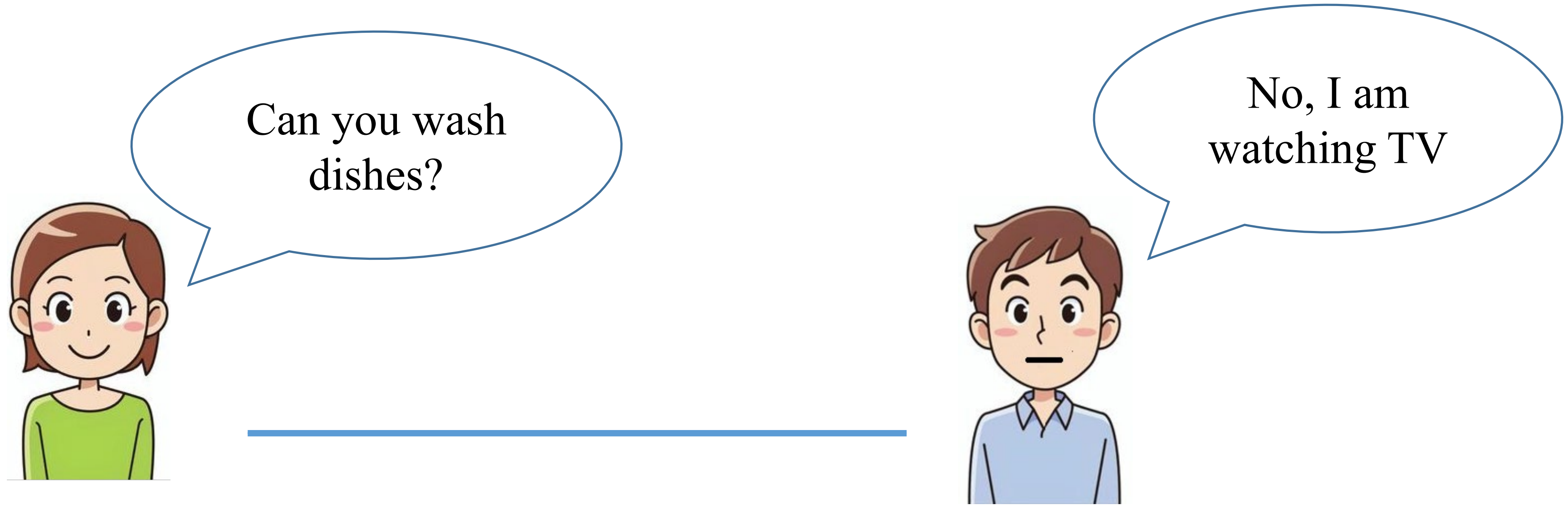
Lawrence Technological University Computer Science

Social Analogy of the Hebbian Learning Rule

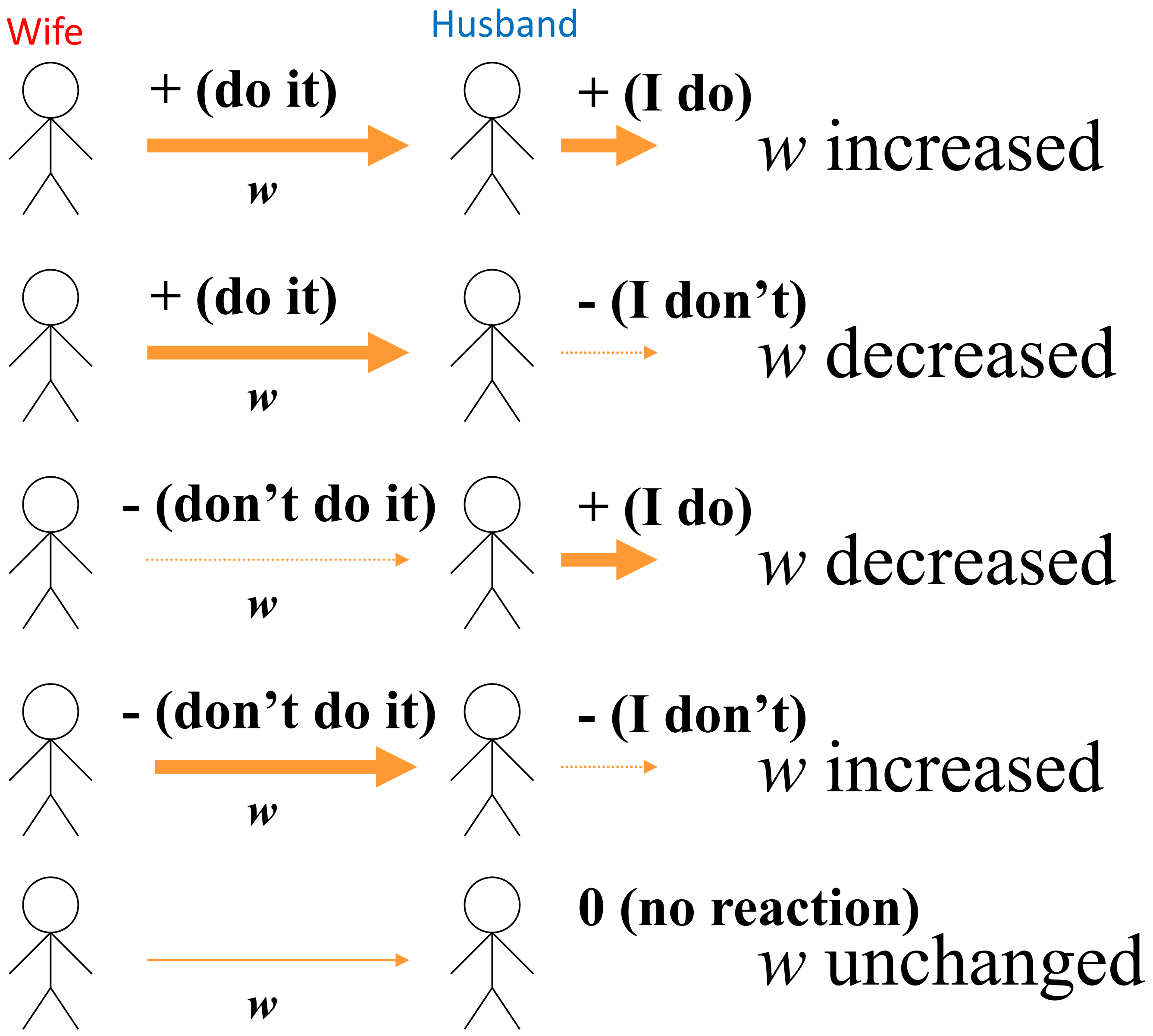


Lawrence Technological University Computer Science

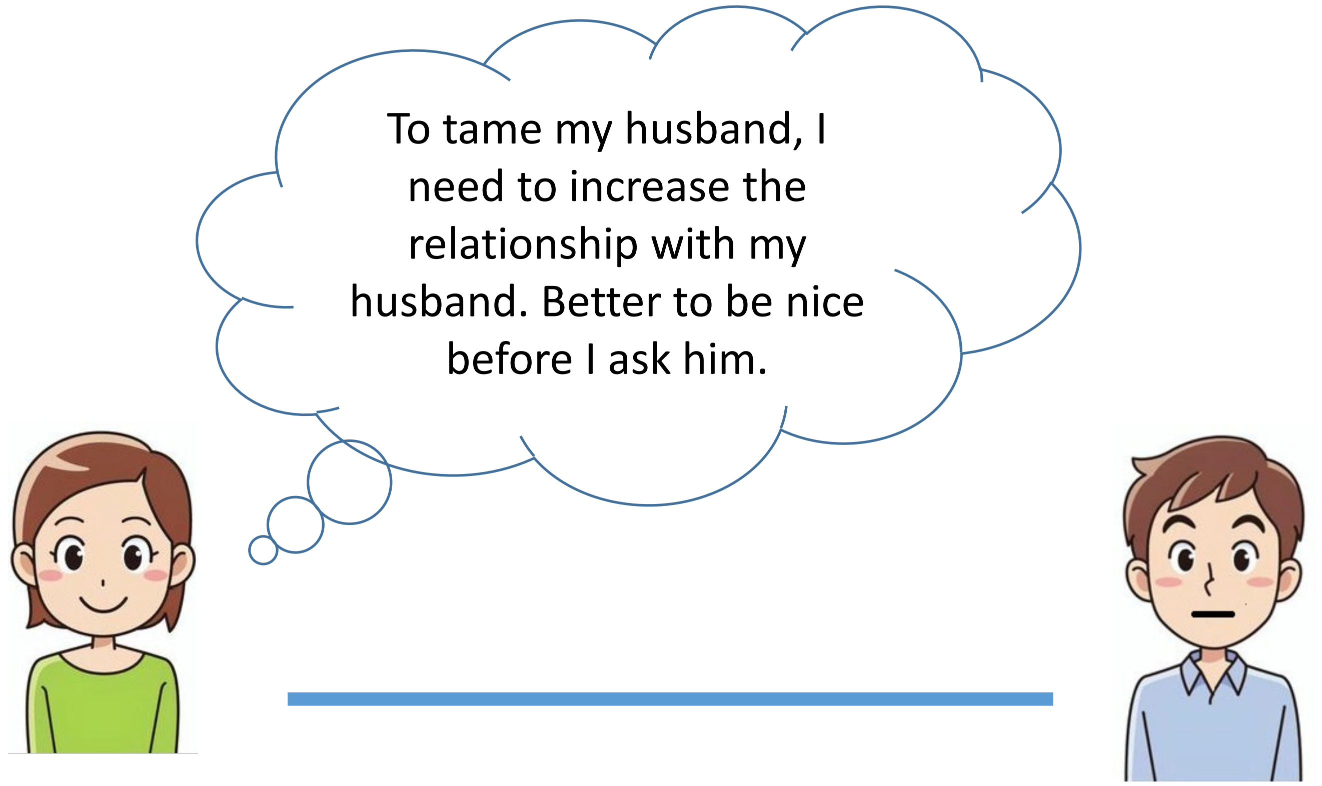
Social Analogy of the Hebbian Learning Rule



Social Analogy of the Hebbian Learning Rule

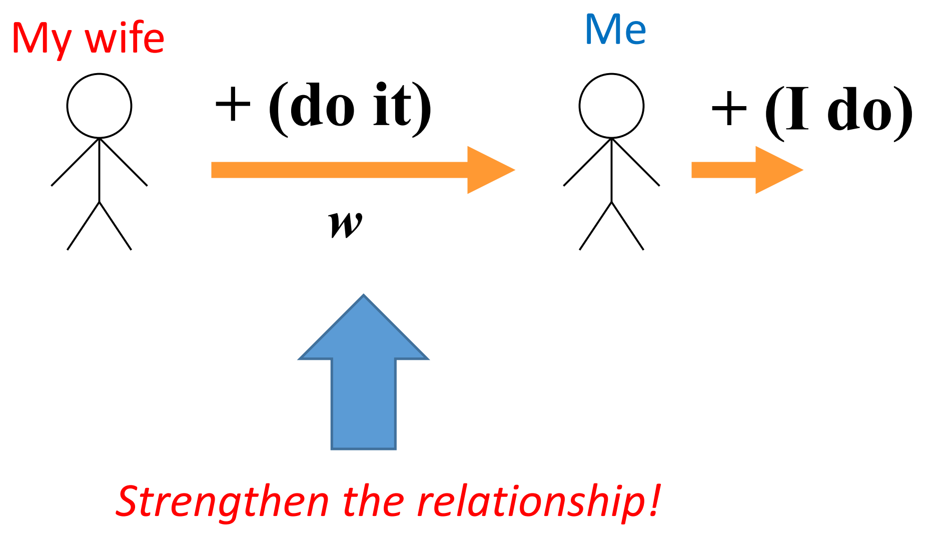


What does she need to do to make me do what she want?

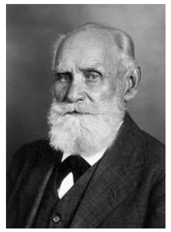

To tame my husband, I need to increase the relationship with my husband. Better to be nice before I ask him.

No way to refuse!

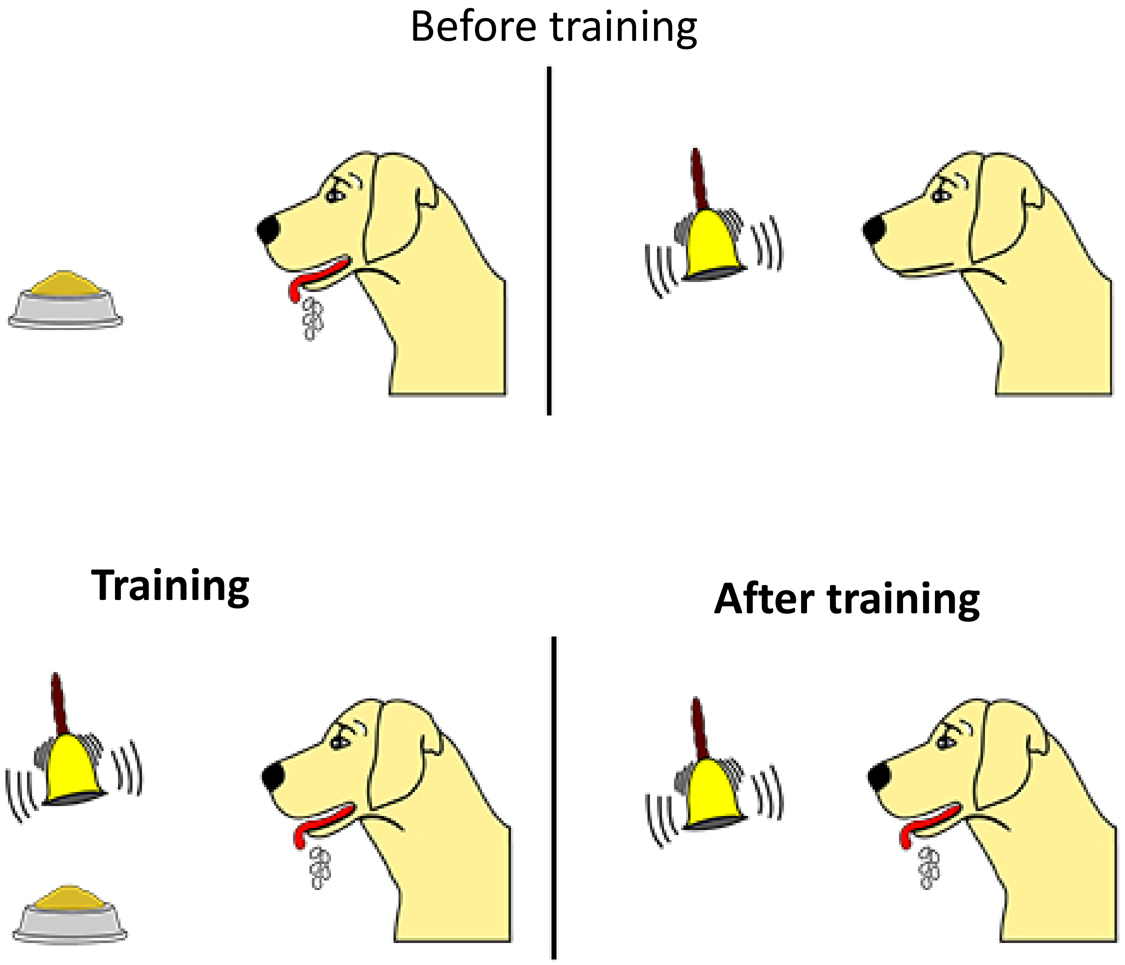
What does she need to do to make me do what she want? Another view



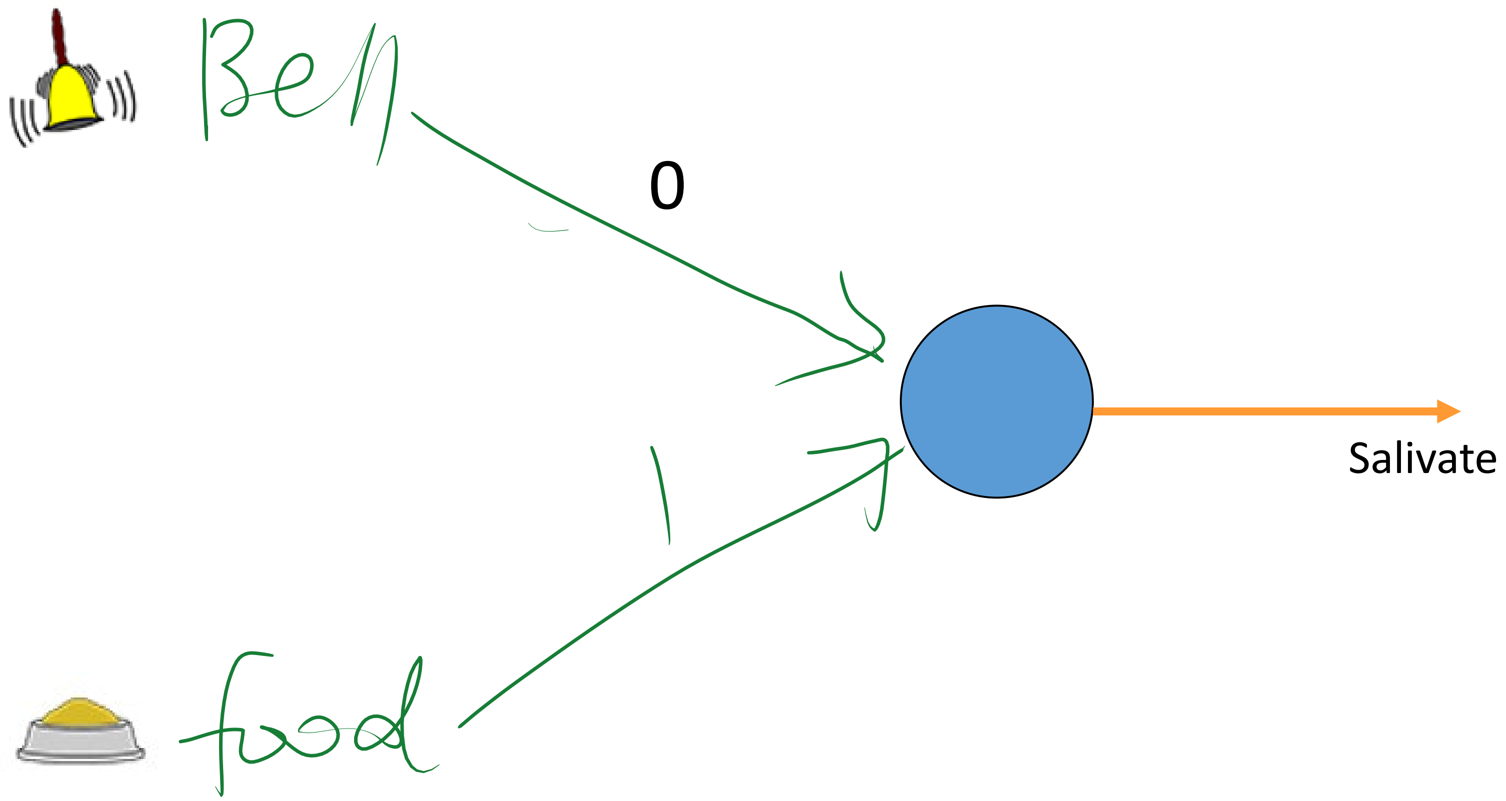
Pavlov's dog



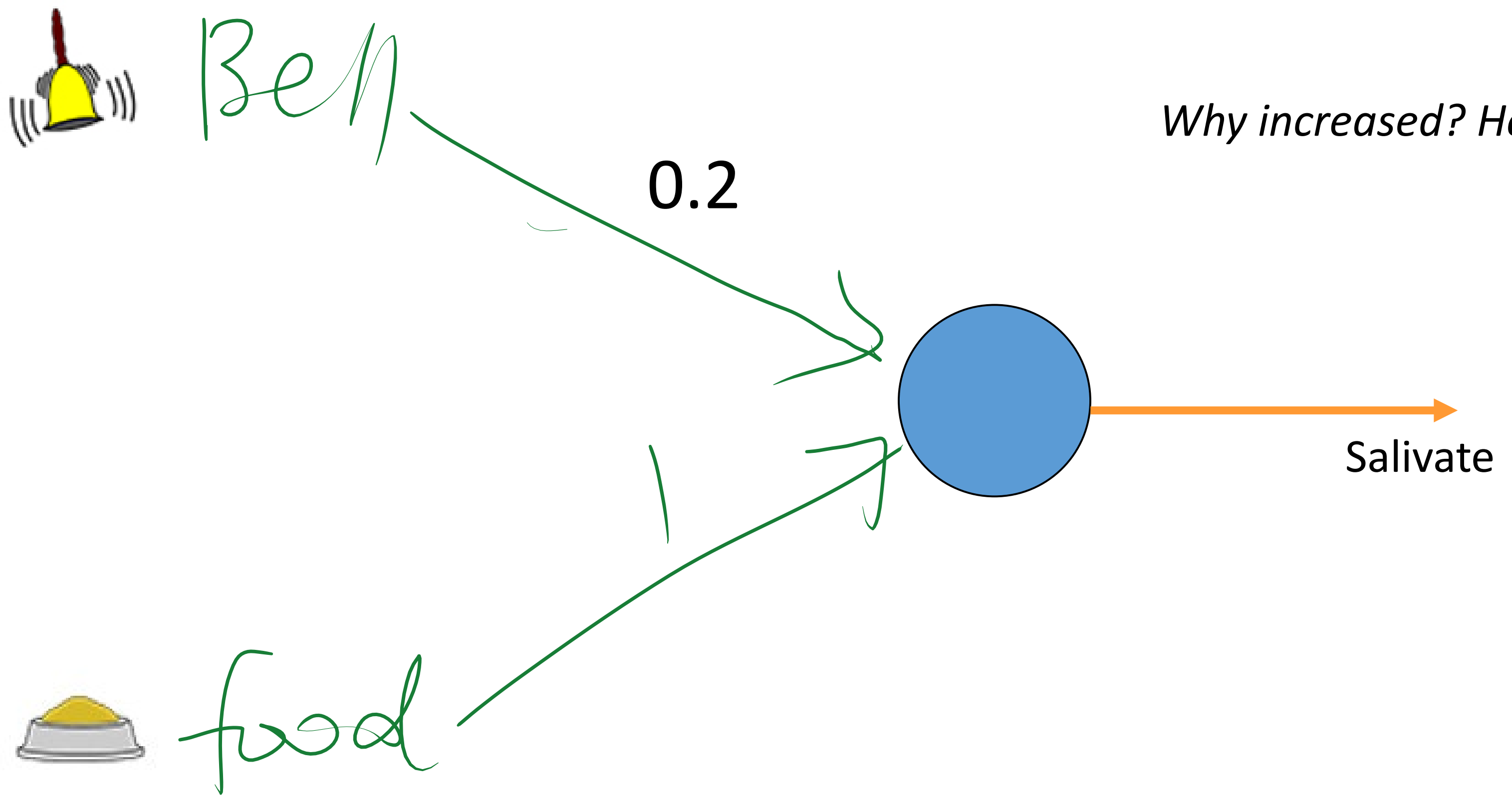
Nobel Prize, 1904



Pavlov's Dog (Before training)

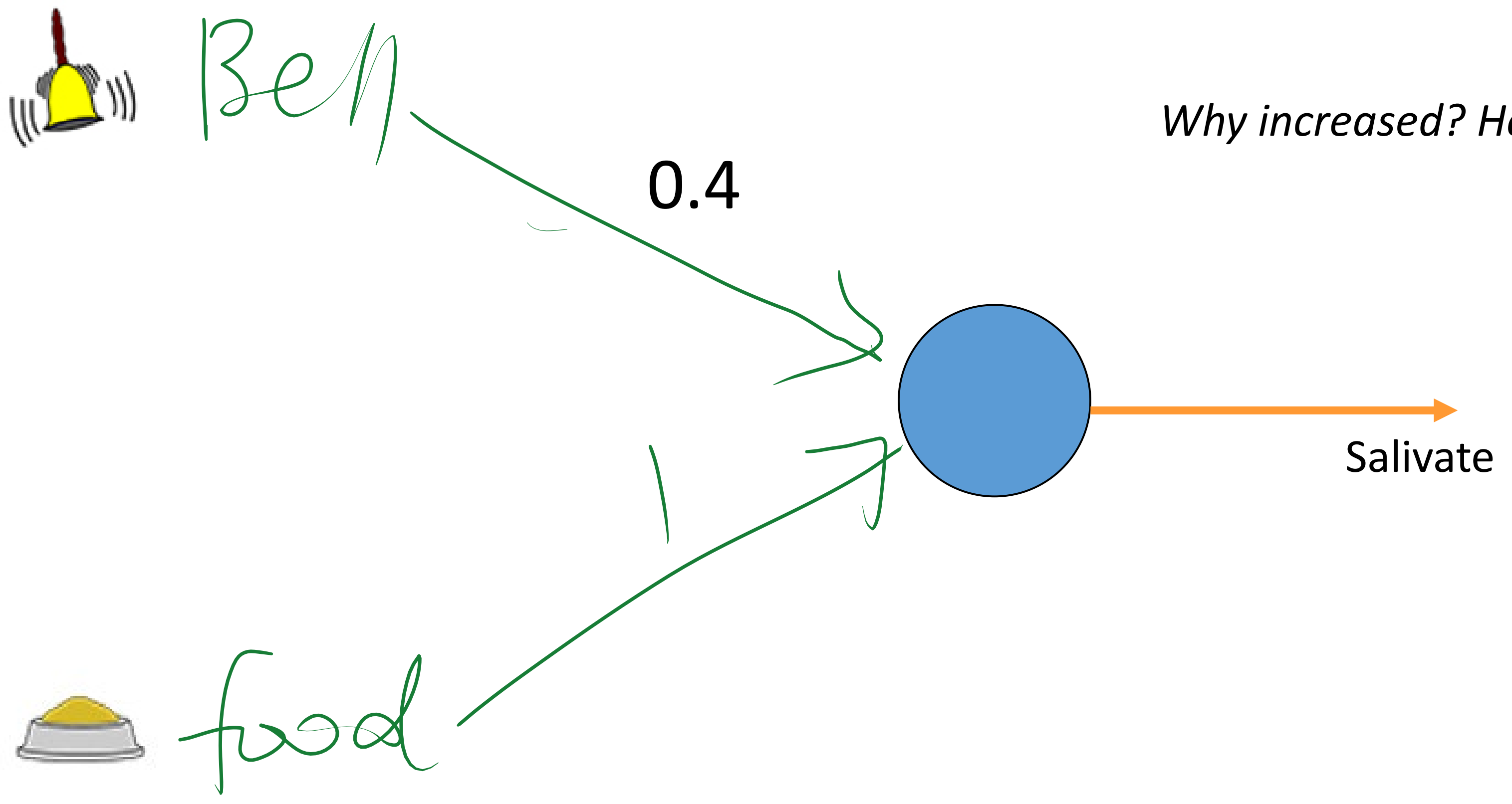


Pavlov's Dog (during training)



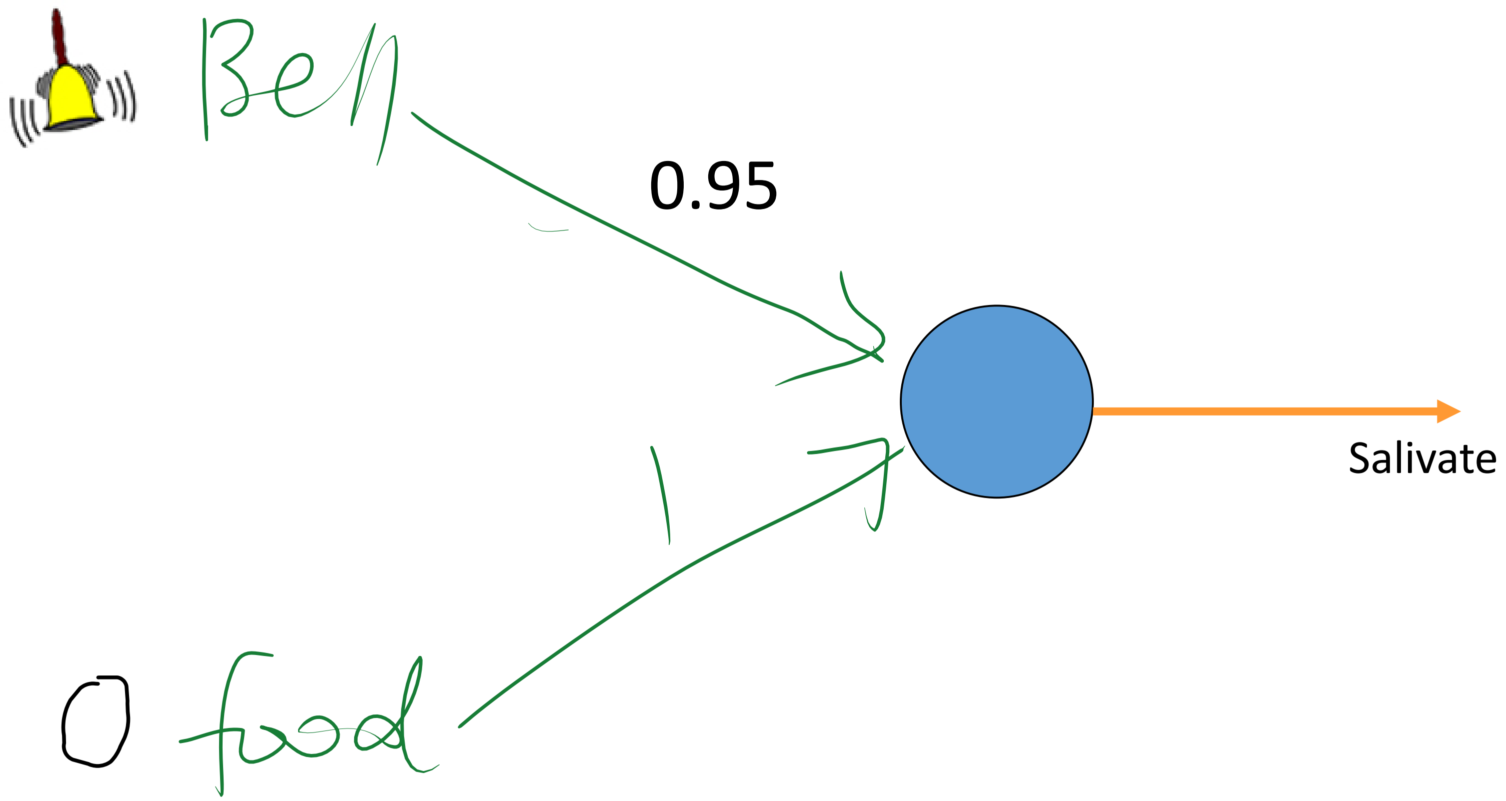
Why increased? Hebbian learning rule.

Pavlov's Dog (during training)



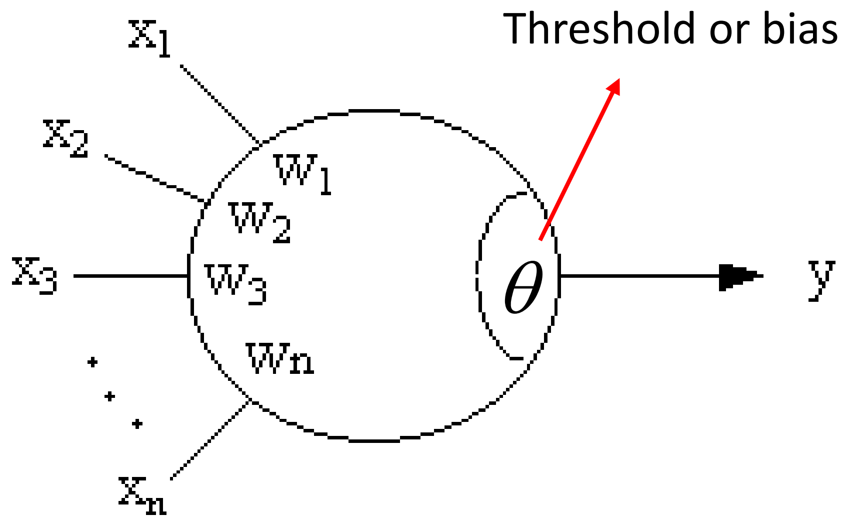
Why increased? Hebbian learning rule.

Pavlov's Dog (after training)



1st Perceptron: A binary classifier

- Rosenblatt: 1957, 1962
- Consists of:
  - Multipliers
  - Adders
  - Thresholds
- The inputs are binary, only 0 and 1
- The output is 0 or 1 depending on whether the weighted sum is greater than the threshold using Stair-step threshold activation function



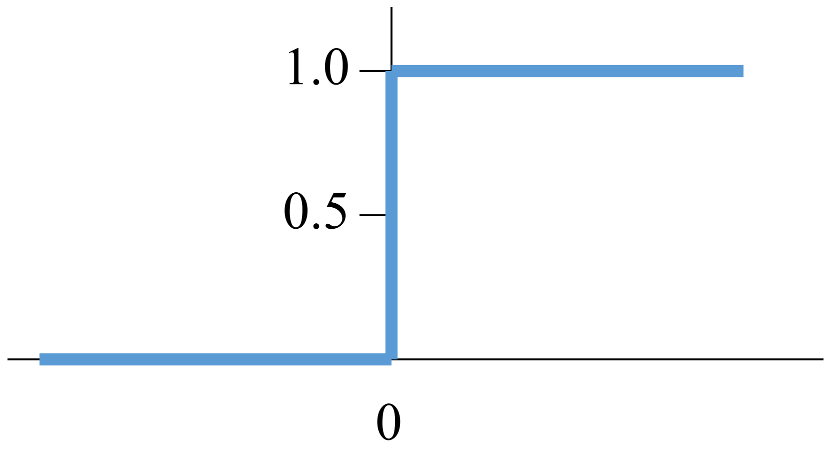
$$wsum = \sum_{i=1}^{n} w_i x_i$$

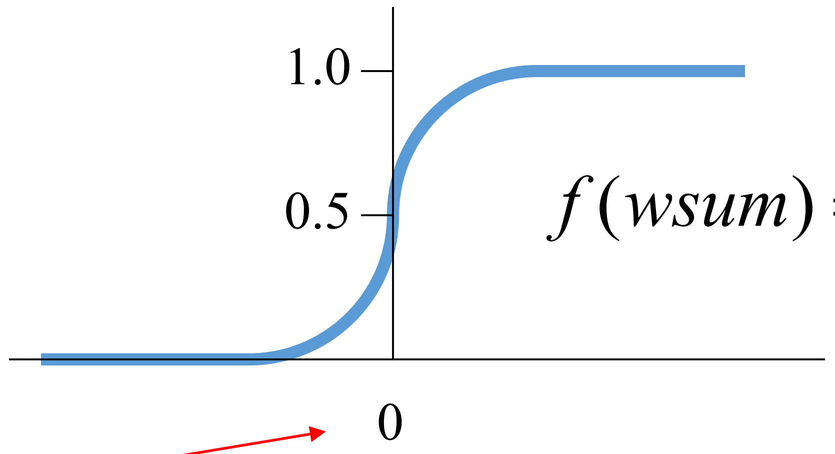
$$y = f(wsum) = \begin{cases} 1 & if\ wsum > \theta \\ 0 & if\ wsum \le \theta \end{cases}$$

Lawrence Technological University Computer Science

Threshold Activation Functions

- Actually, the stair-step threshold function is not suited for gradient descent, because activation function requires performance to be a smooth function of the weights.
- Squashed S (Sigmoid) activation function is widely used.



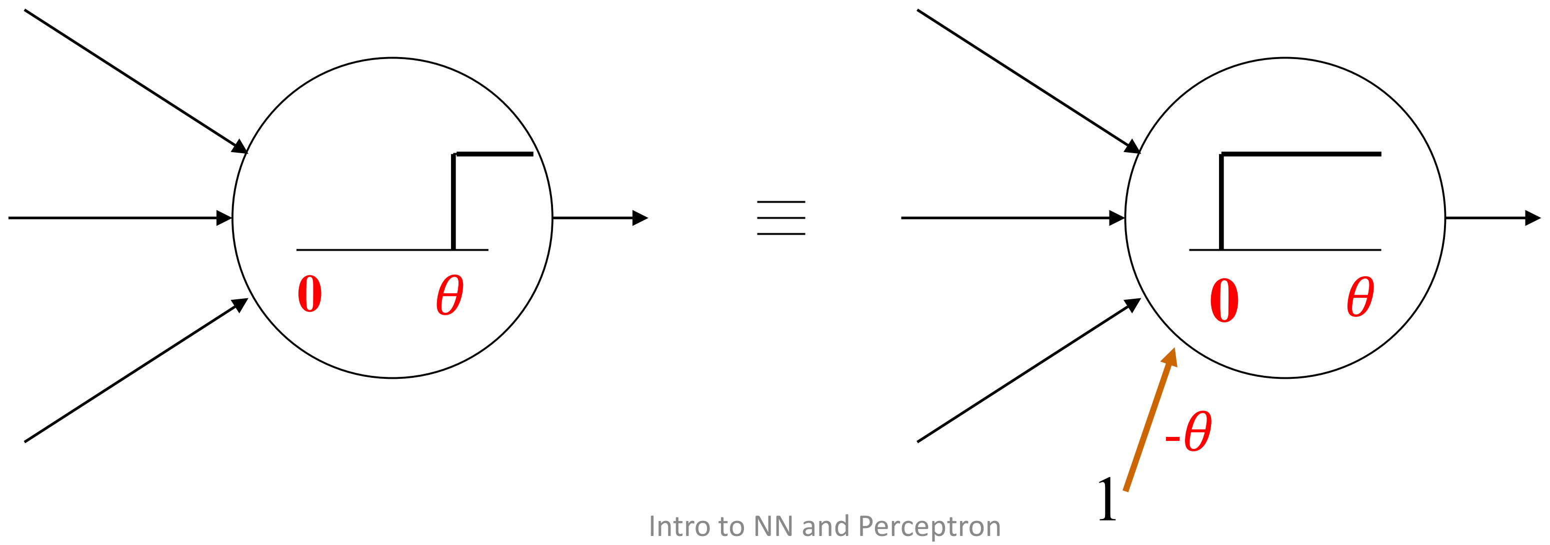


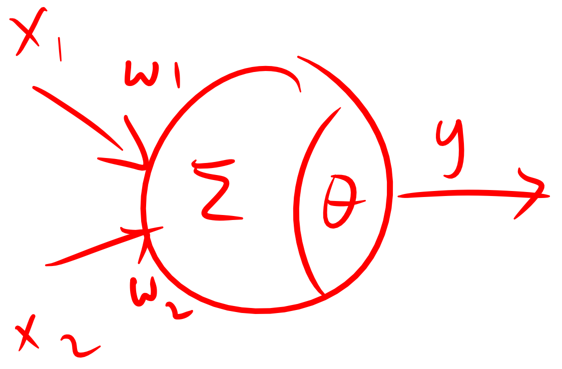
$$f(wsum) = \frac{1}{1+e^{-wsum}}$$

Threshold (Bias) is zero

Removing non-zero threshold value

- Non-zero threshold can be eliminated.
- A nonzero-threshold neuron is computationally equivalent to a **zero-threshold** neuron with an extra link held at 1 (not −1, in case of basic perceptron). The $-\theta$ value becomes the connecting weight's value.

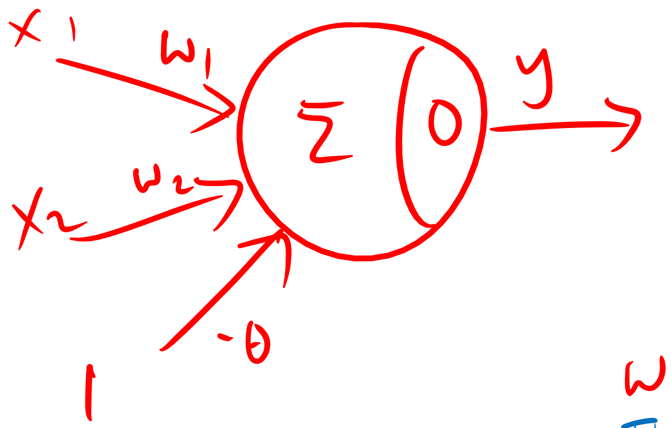




$$y = \begin{cases} 1 & \text{if } (x_1 \cdot w_1 + x_2 \cdot w_2) > \theta \\ 0 & \text{if } (x_1 \cdot w_1 + x_2 \cdot w_2) \le \theta \end{cases}$$

$\Downarrow$

also called as bias $\swarrow$ w_3

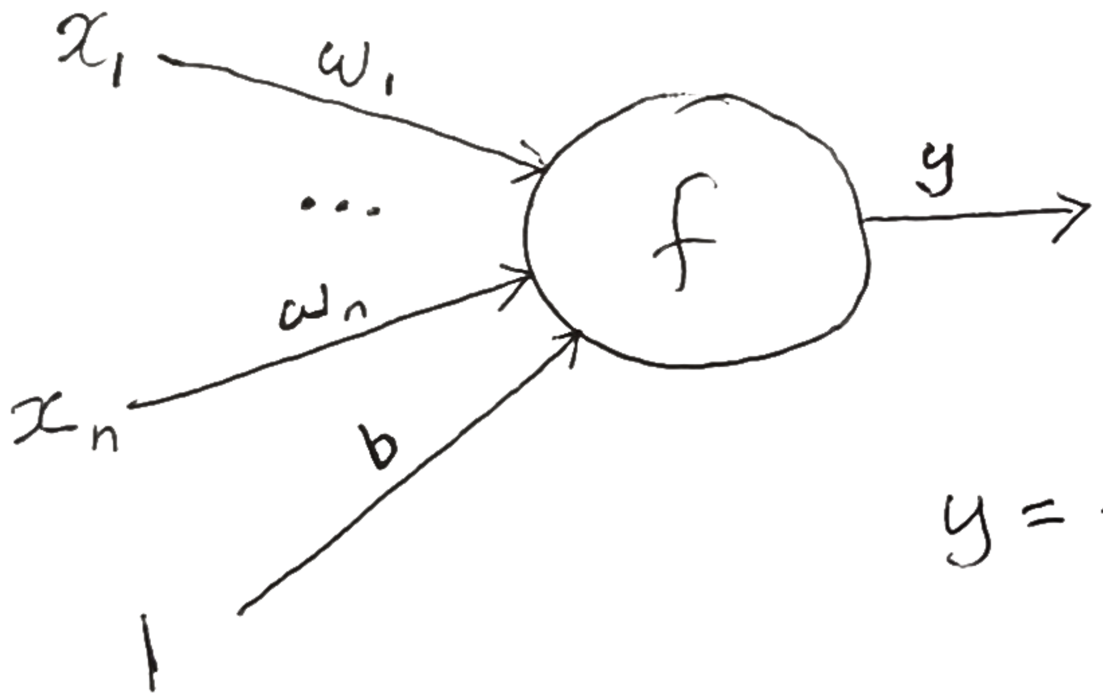


$$y = \begin{cases} 1 & \text{if } (x_1 w_1 + x_2 w_2 + 1 \cdot \underbrace{(-\theta)}) > 0 \\ 0 & \text{if } (x_1 w_1 + x_2 w_2 + 1 \cdot \underbrace{(-\theta)}) \le 0 \end{cases}$$

We do not care the sign of the θ.

The goal is to find w_1, w_2, and w_3 that satisfy the inequalities for all samples.

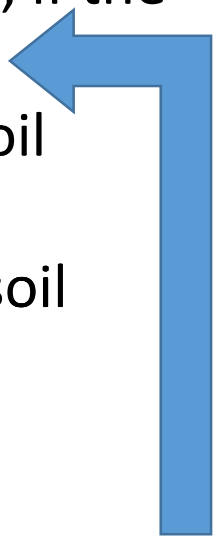
To calculate output *y* of a neuron with activation function *f* and threshold bias *b*



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

How to train the Perceptron?
How to adjust the weight values?

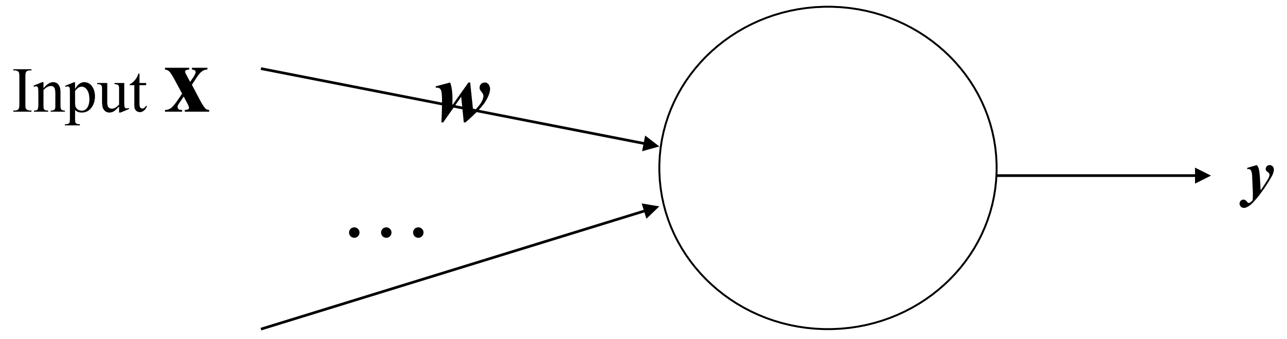
- 3 cases of watering:
  - Keep the current amount of watering, if the soil humidity is good
  - Increase the current amount, if the soil humidity becomes too dry
  - Decrease the current amount, if the soil humidity becomes too wet



We (as supervisors) know the desired outcome!!

How to train the Perceptron?
How to adjust the weight values?

- 3 cases of watering
  - Keep the current amount of watering, if good
  - Increase the current amount, if too dry
  - Decrease the current amount, if too wet



Keep, Decrease, or Increase ***w*** based on ***y***
Hebbian Learning Rule!

Classes of learning methods

- *Corrective* **Supervised**: a situation in which sample (input, output) pairs of the function to be learned can be perceived or given. Learning with a teacher who provides the input data as well as the desired solution
- **Reinforcement**
  - in the case of the agent acts on its environment, it receives *some* evaluation of its action
  - Example: Mobile phone users to obtain good reception. We move around (explore) with the phone while monitoring its signal strength indicator or by repeating "Do you hear me now?" We are not given examples of correct behavior.
- **Unsupervised**: Learn to produce identical response to similar activation patterns on the input units without an external guidance.

How to update weights?
A simple Perceptron learning rule

$$\mathbf{w}'_k = \begin{cases} \mathbf{w}_k & \text{if } \overset{\text{input}}{\mathbf{x}_k} \text{ is correctly classified by } \mathbf{w}_k \\ \mathbf{w}_k - \mathbf{x}_k & \text{if false positive} \\ \mathbf{w}_k + \mathbf{x}_k & \text{if false negative} \end{cases}$$

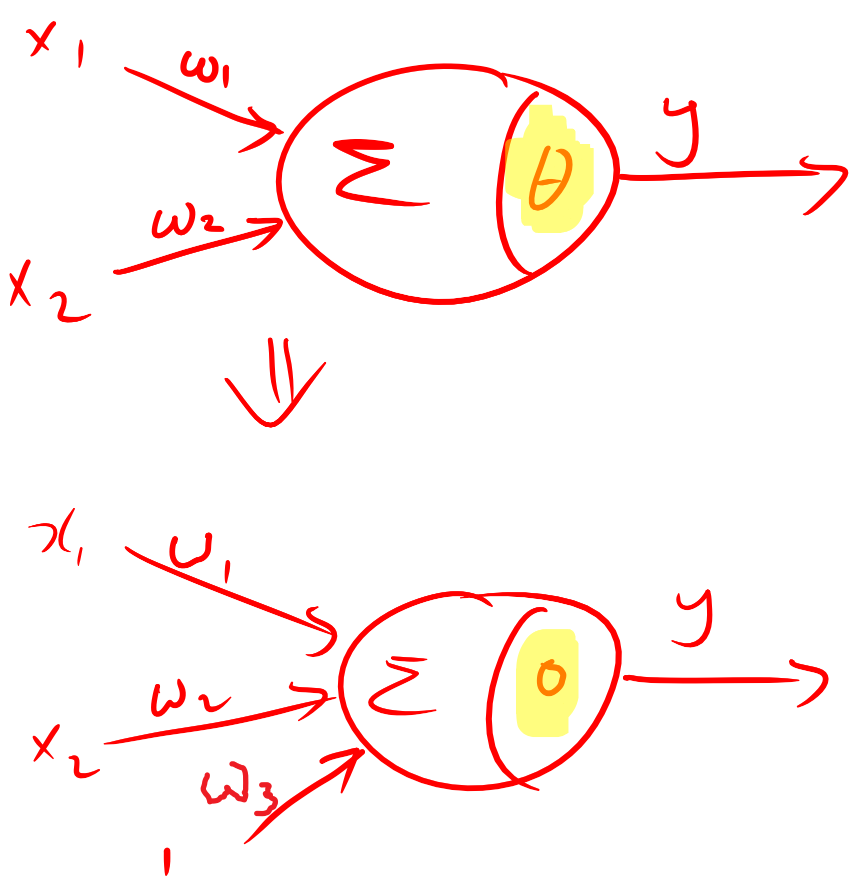
Corrective Supervised learning!

Perceptron Learning Example: OR function

Truth table

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

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



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

Let's start with an initial weight value $W_0 = \begin{bmatrix}0\\0\\0\end{bmatrix}$

1st phase(epoch)

$$W_0^t \cdot X_1 = [0\ 0\ 0] \cdot \begin{bmatrix}0\\0\\1\end{bmatrix} = 0$$ ok

$$W_0^t \cdot X_2 = [0\ 0\ 0] \begin{bmatrix}0\\1\\1\end{bmatrix} = 0$$ (false neg.)

So $W_1^t = W_0^t + X_2 = [0\ 1\ 1]$

$$W_1^t \cdot X_3 = [0\ 1\ 1] \begin{bmatrix}1\\0\\1\end{bmatrix} = 1$$ (ok)

$$w_1^t \cdot x_4 = [0\ 1\ 1] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \boxed{2} = \text{yes}$$

1 ok

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

2nd phase

$$w_1^t \cdot x_1 = [0\ 1\ 1] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 1$$

false positive

$$W_2 = W_1 - X_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\mathbf{w}'_k = \begin{cases} \mathbf{w}_k & \text{if } \mathbf{x}_k \text{ is correctly classified by } \mathbf{w}_k \\ \mathbf{w}_k - \mathbf{x}_k & \text{if false positive} \\ \mathbf{w}_k + \mathbf{x}_k & \text{if false negative} \end{cases}$$

$$w_2^t \cdot x_2 = [0\ 1\ 0] \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 1 \quad \text{ok}$$

$$W_3^t \cdot X_3 = (0\ 1\ 0)\begin{pmatrix}1\\0\\1\end{pmatrix} = 0$$ false neg.

$$W_3 = W_2 + X_3 = \begin{bmatrix}0\\1\\0\end{bmatrix} + \begin{bmatrix}1\\0\\1\end{bmatrix} = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

$$W_3^t \cdot X_4 = (1\ 1\ 1)\begin{bmatrix}1\\1\\1\end{bmatrix} = 3$$ ok

3rd phase

$$W_3^t \cdot X_1 = (1\ 1\ 1)\begin{bmatrix}0\\0\\1\end{bmatrix} = 1$$ false positive

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

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

$$W_4 = W_3 - X_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$W_4^t \cdot X_2 = [1\ 1\ 0]\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 1$$

$$W_4^t \cdot X_3 = (1\ 1\ 0)\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 1$$

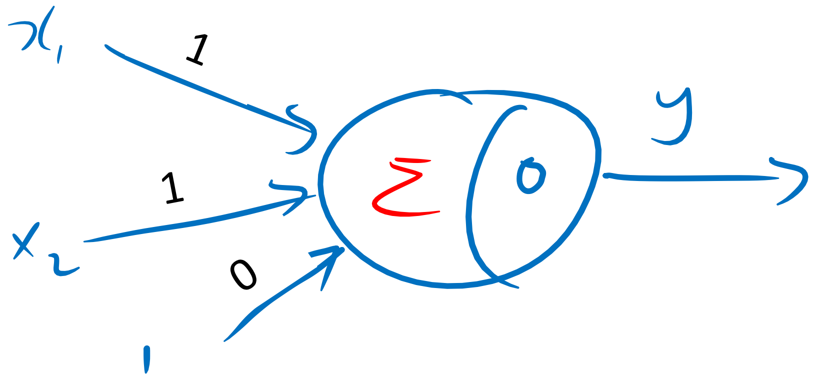
$$W_4^t X_4 = [\underline{1\ 1}\ 0]\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 2 \quad \text{ok}$$

$$W_4 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$ works for all 4 samples!

Let's verify

$W_4 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ works for all 4 samples!

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



Perceptron training algorithm (general, real valued)

- Generate random weights, usually between –0.5 & 0.5
- Repeat the following until $\boldsymbol{w}$ is found that *correctly* classifies ***all*** the training data:

Present an item (**X**) of training data samples. The weights are modified according to the following:

$$w_i = w_i - (\alpha \cdot e \cdot x_i)$$

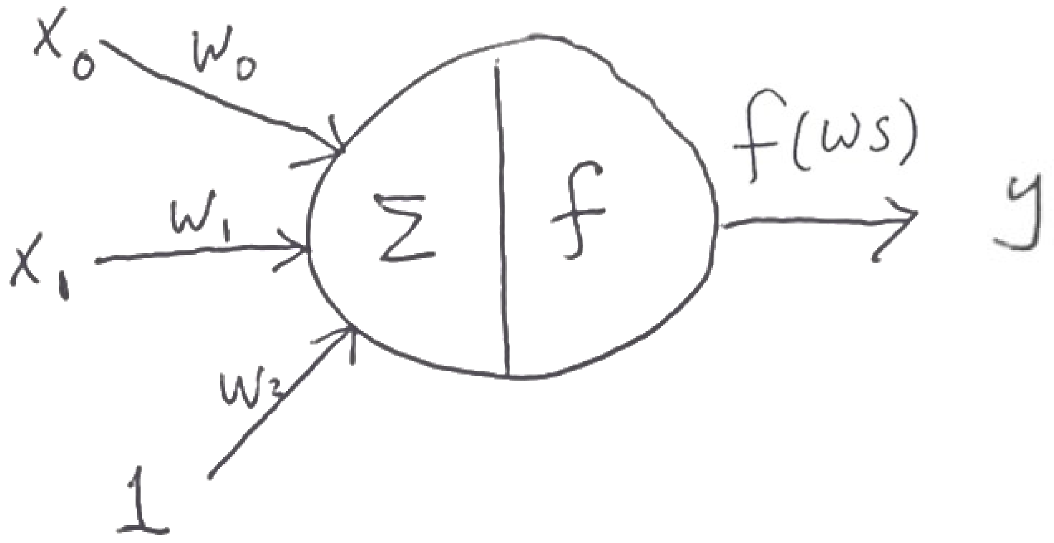
α is the learning rate, between 0 and 1

e = (**perceptron output - target value**)

- 0, if output is correct: no weight change
- Positive, if output is too high: decrease w
- Negative, if output is too low: increase w

Sample perceptron code for the OR gate is posted:
Perceptron_stairStep.ipynb

X_0	X_1	1	y
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1



$$f(ws) = \begin{cases} 1 & ws > 0 \\ 0 & \text{else} \end{cases}$$

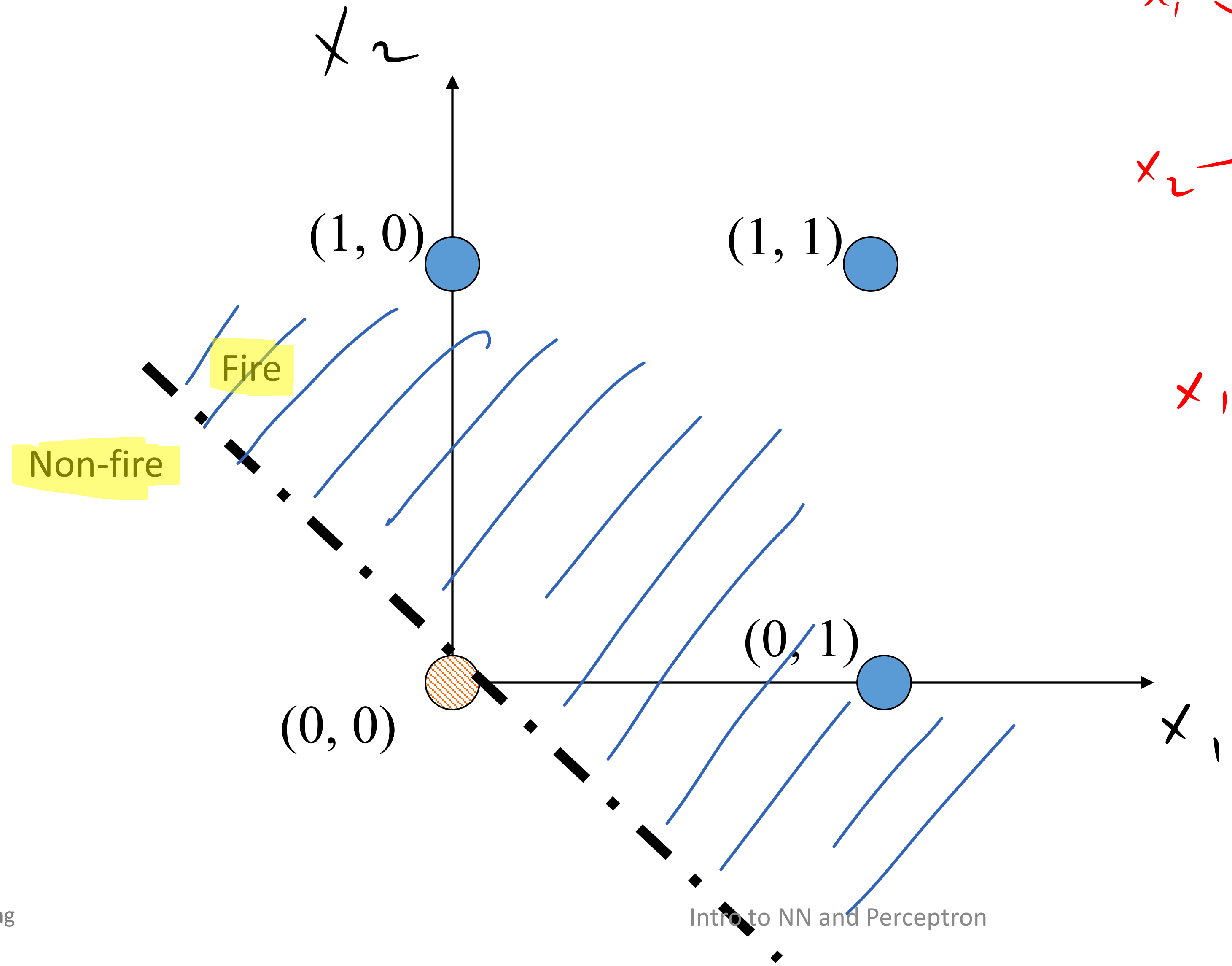
$$ws = \sum_{i=0}^{n-1} x_i \cdot w_i + w_n$$

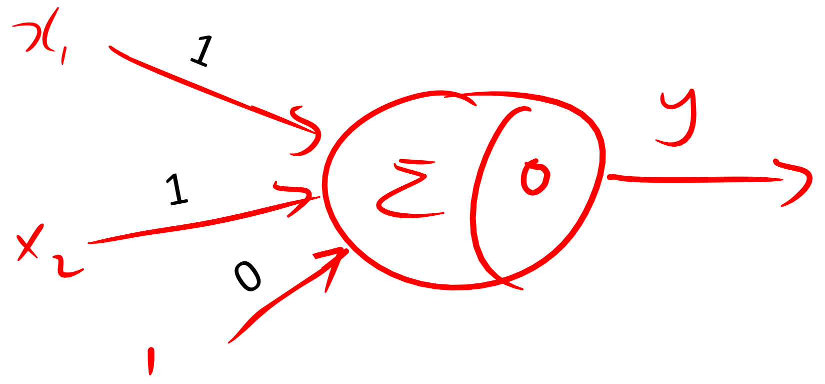
Sample code for the OR gate:

Perceptron_stairStep.ipynb

Compare this program with OR_function.ipynb using Keras, later

OR: Linearly Separable Problem

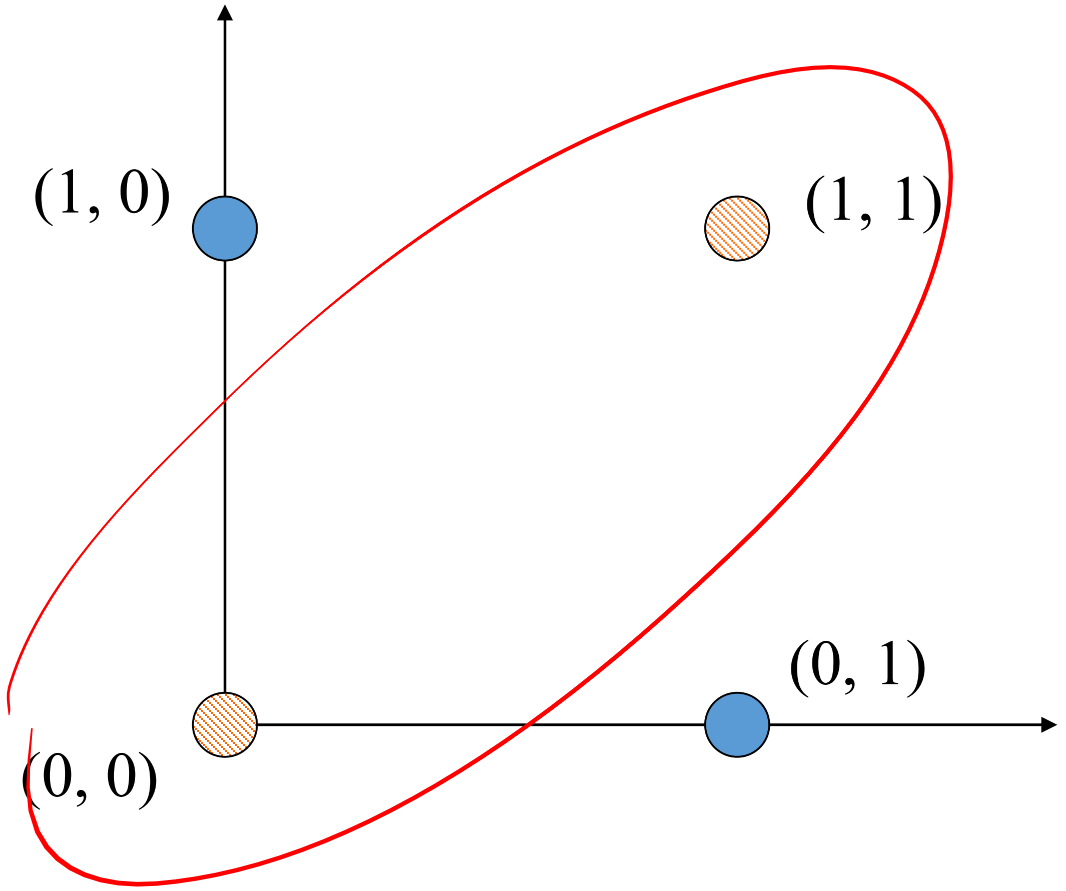




$x_1 + x_2 + 0 > 0$ Then, fire

$x_2 > -x_1$

XOR: Linearly Non-separable Problem



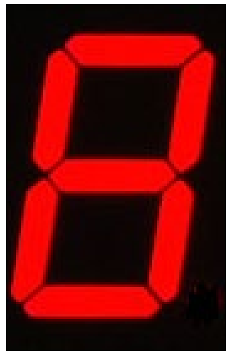
1969 MIT Media Lab Dr. Papert: "Perceptron cannot solve linearly non-separable problems"

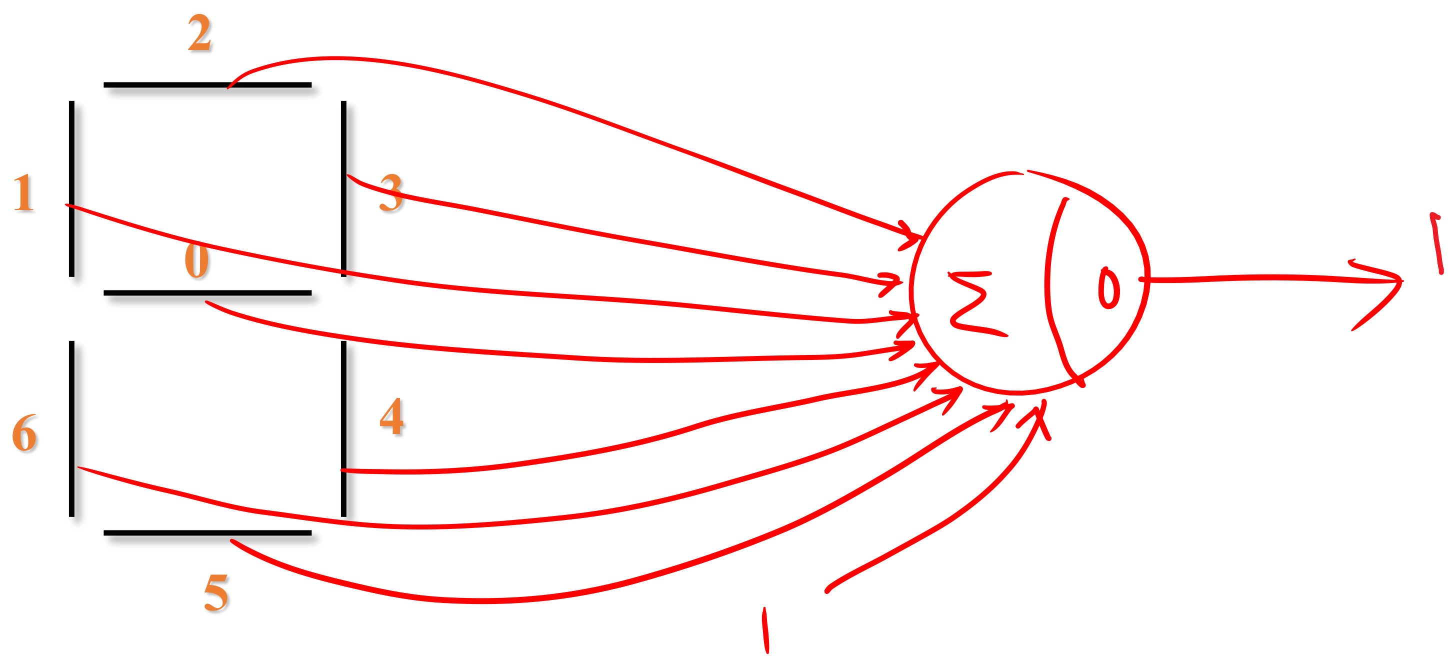
x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Sample Keras codes with multiple neurons for the XOR function: **XOR.ipynb** will be covered later.

Perceptron Recognizing 7 Segment Display

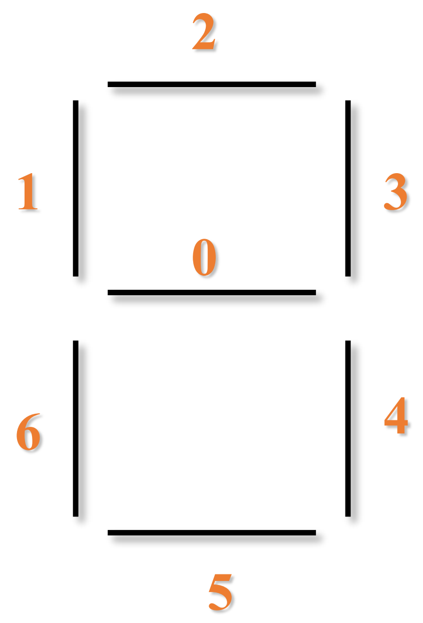
- To train a Perceptron to identify “8”, as an example
- Test it with 0 ~ 9

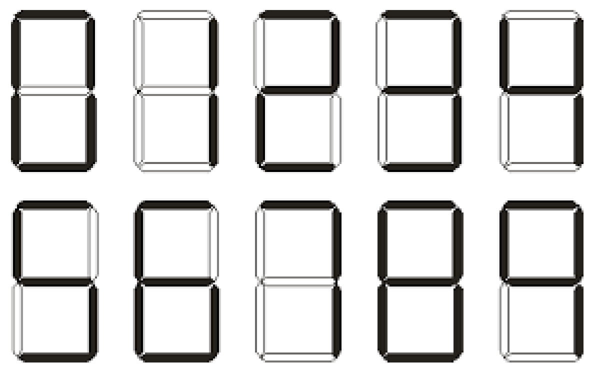




Training Data to recognize “8”

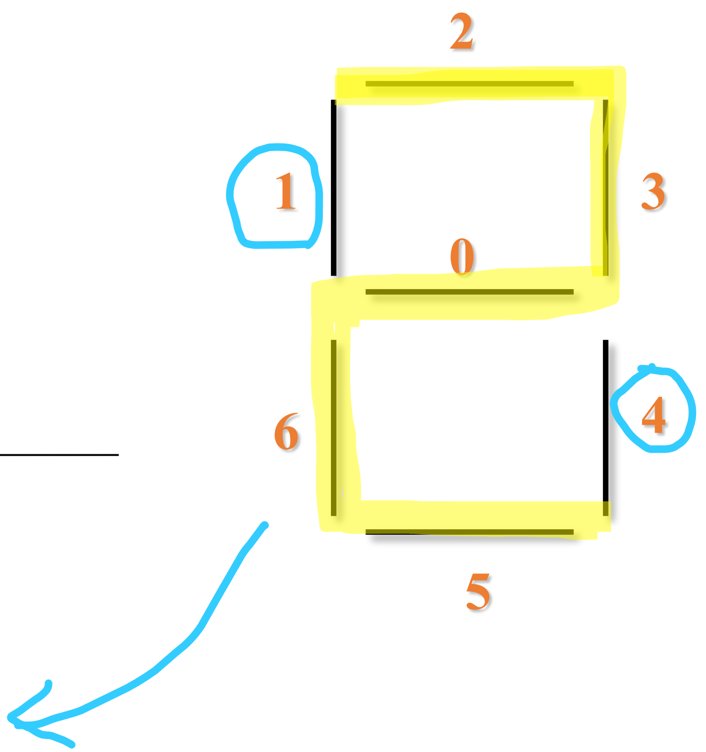
	0	1	2	3	4	5	6	1	**y**
0:	0	1	1	1	1	1	1	1	**0**
1:	0	0	0	1	1	0	0	1	**0**
2:					?				
				…					
8:	1	1	1	1	1	1	1	1	**1**
9:	1	1	1	1	1	0	0	1	**0**

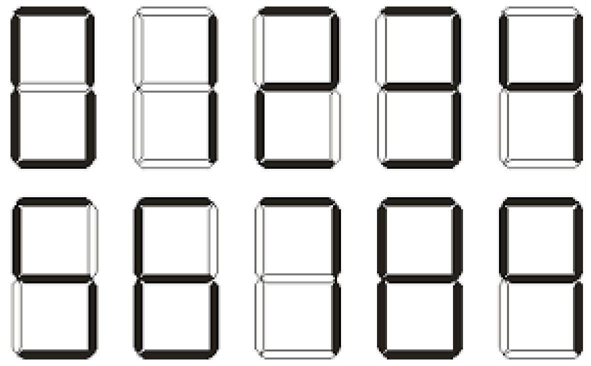




Training Data for “8”

	0	1	2	3	4	5	6	1	y
0:	0	1	1	1	1	1	1	1	**0**
1:	0	0	0	1	1	0	0	1	**0**
2:	1	**0**	1	1	**0**	1	1	1	**0**
…									
8:	1	1	1	1	1	1	1	1	**1**
9:	1	1	1	1	1	0	0	1	**0**





1st epoch

$$W_0^t = [0\ 0 \cdots 0]$$

$$W_0^t \cdot X_0 = [0\ 0 \cdots 0] \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = 0 \quad \text{ok}$$

$$W_0^t \cdot X_1 = [0\ 0 \cdots 0] \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} = 0 \quad \text{ok}$$

$$W_0^t \cdot X_2 = [0 \quad 0] \begin{bmatrix} \\ \\ \end{bmatrix} = 0 \quad \text{ok}$$

$$\vdots$$

$$W_0^t \cdot X_8 = [0 \cdots 0] \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = 0 \quad \text{false neg.}$$

$$W_1 = W_0 + X_8$$

$w_1 = [1\,1\,1\,1\,1 \cdot \quad]$

$w_1^t \cdot X_9 = [1\,1\,1 \cdot \cdot \,] \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = 7$ false pos.

$W_2 = W_1 - \underline{X_9} = [0\,0\,0\,0\,0\,0\,1\,0]$

<u>2nd epoch</u>

$W_2^t \cdot X_0 = [0\,0\,0\,0\,0\,0\,1\,0] \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = 1$

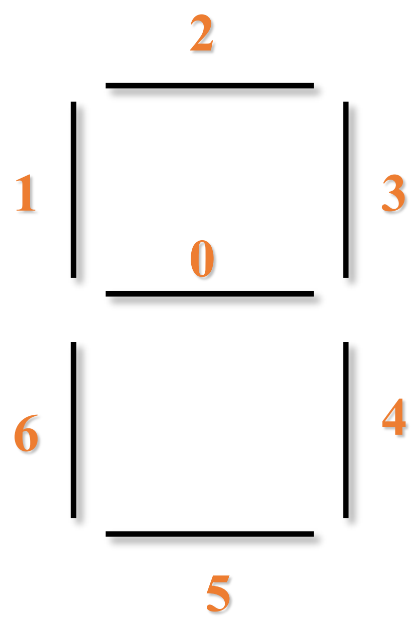
false pos.

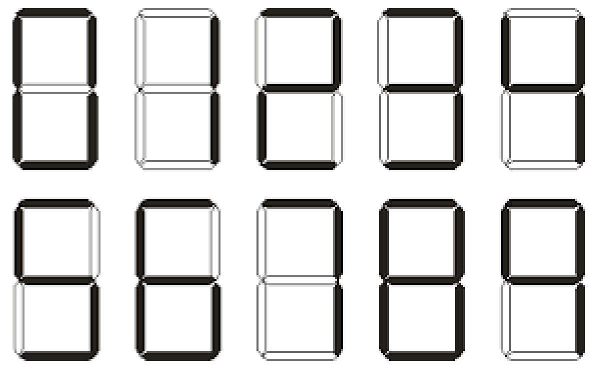
$W_3 = W_2 - X_0$

$\cdots$

Old HW Assignment

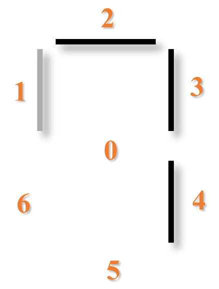
Write an *.ipynb program to train a Perceptron to identify “**7**” of the 7 segment display. Use digits & indices ***exactly*** as the figure below to prepare training samples in a Numpy array.



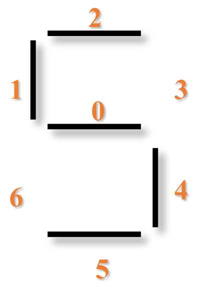


Test Cases to use for the assignment

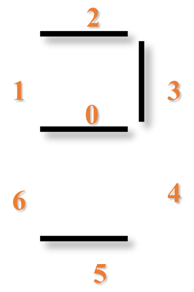
[0.0, **0.5**, 1.1, 1.0, **1.2**, 0.0, 0.0, 1.0] →



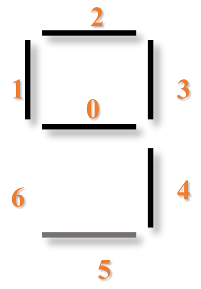
[**0.9**, 1.1, 1.0, 0.0, **1.1**, 1.0, 0.0, 1.0] →



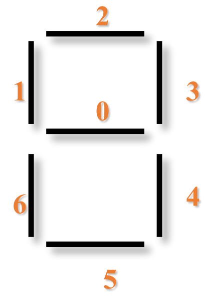
[1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0] →



[**0.9**, 1.0, 1.0, 1.0, 1.1, 0.0, **0.5**, 1.0] →

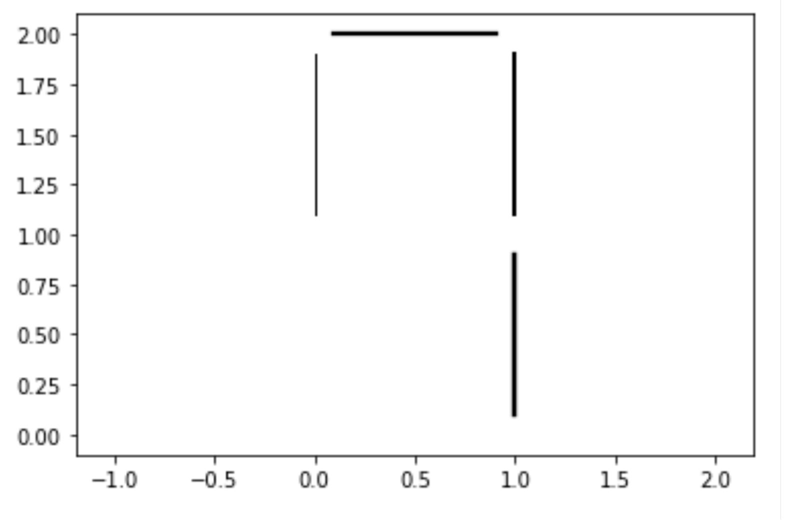


[**1.2**, 1.0, 1.0, **1.1**, 1.0, 1.0, **0.8**, 1.0] →



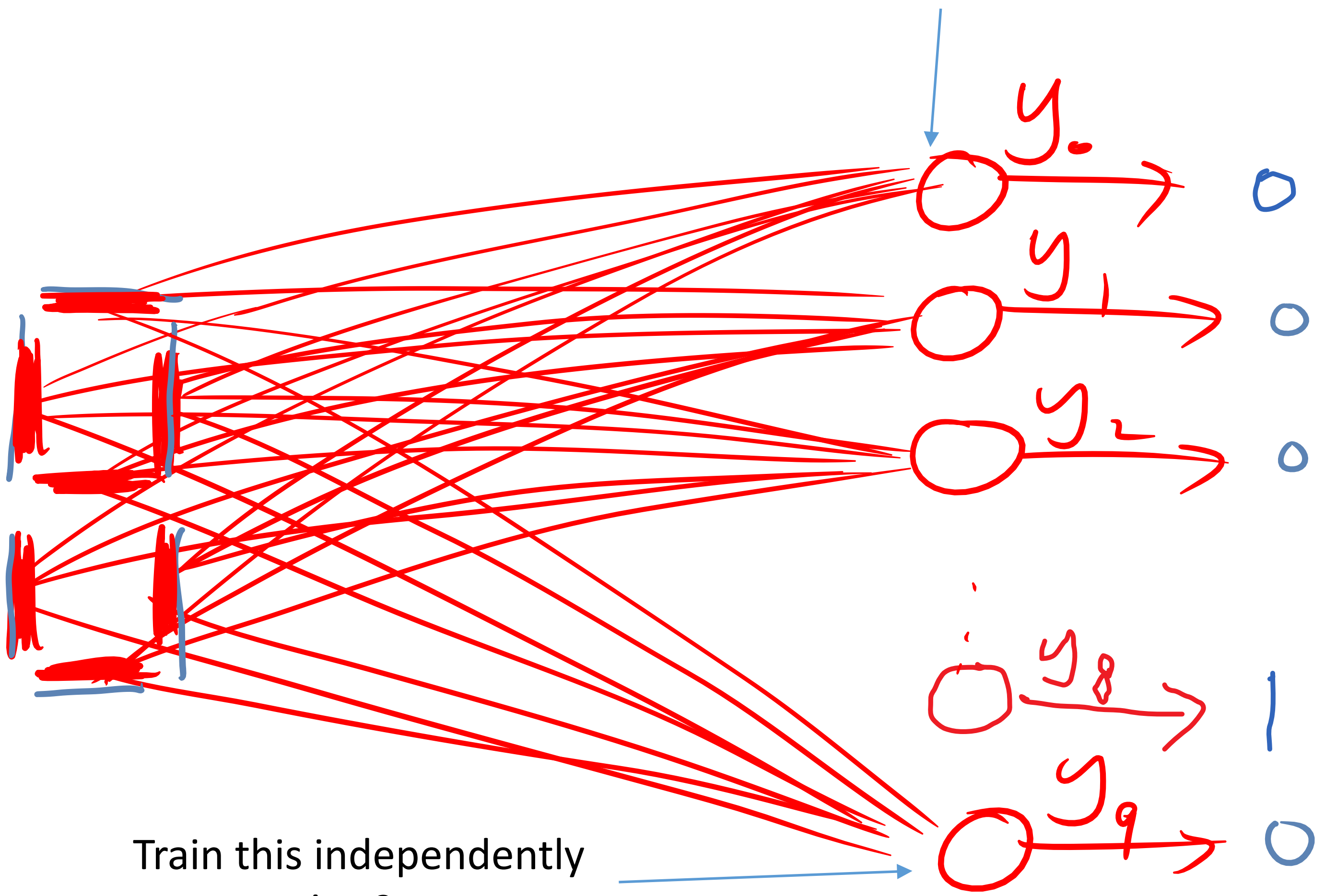
Suggested Improvements for the assignment (old)

- Change to f string to print
- Format nicely for human readers
- For all digits
- Display the digit: See Perceptron7seg_for7_HW3_sol.ipynb



Using 10 Perceptrons to recognize 10 digits

Train this independently
to recognize 0.



Train this independently
to recognize 9.

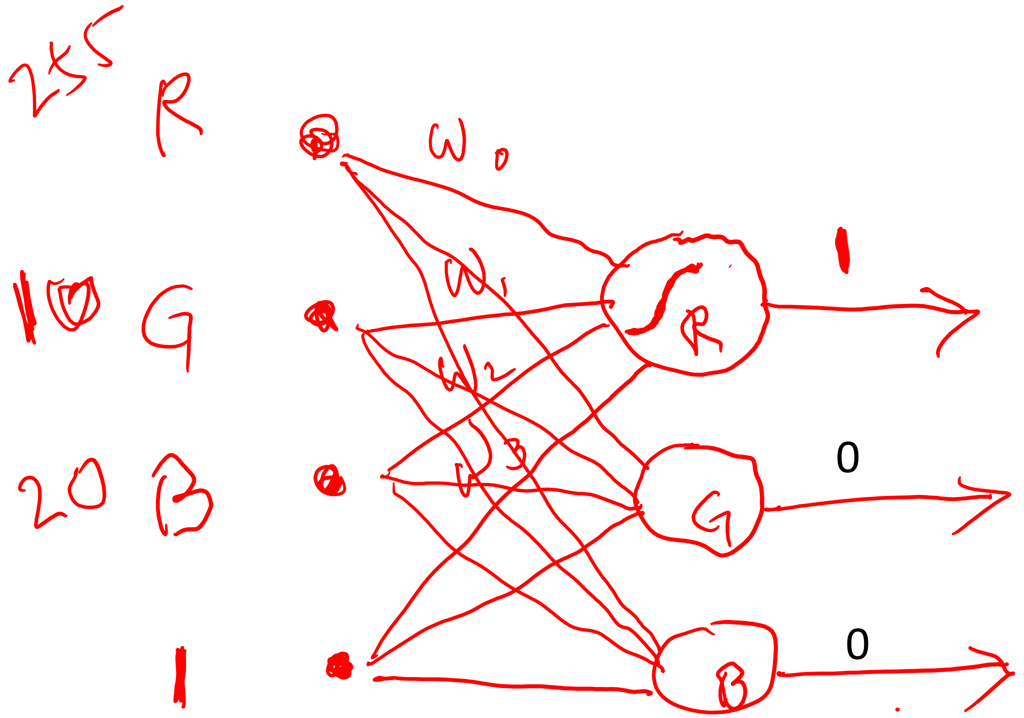
Self-Exercises

Ex1. Design a Perceptron for “AND” function

Ex2. Design a Perceptron to recognize **red** or not, when R, G, B values are given; Design also a sample training data set

Ex3. Design a network of multiple Perceptrons that classifies 3 colors (R, G, and B); Design also a sample data set for training

Ex. 3



R	G	B	1	y
255	0	0	1	1
252	10	20	1	1
90	255	0	1	0
11	12	250	1	0
		…		

Training dataset for R

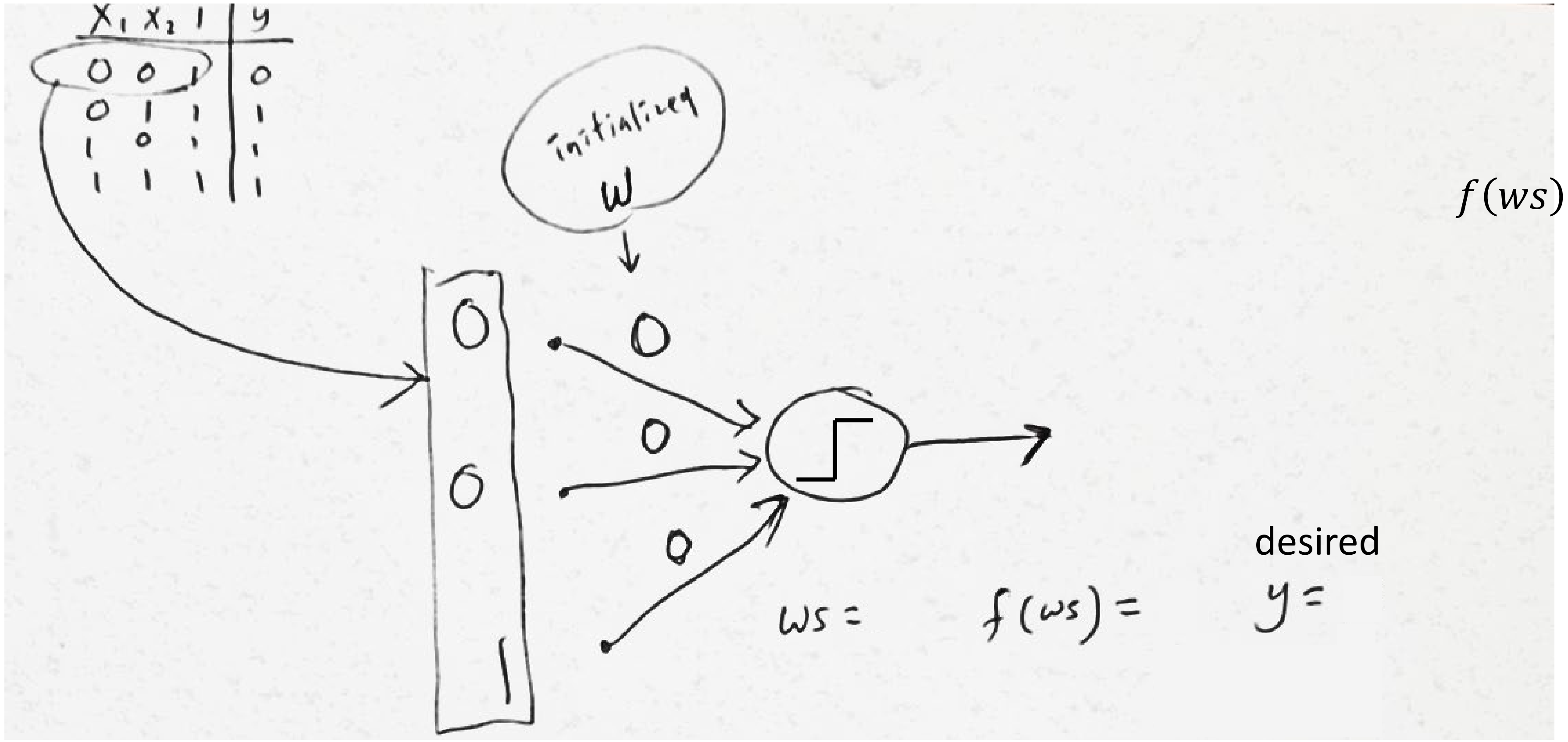
R	G	B	1	y
255	0	0	1	0
252	10	20	1	0
90	255	0	1	1
11	12	250	1	0
		…		

Training dataset for G

R	G	B	1	y
255	0	0	1	0
252	10	20	1	0
90	255	0	1	0
11	12	250	1	1
		…		

Training dataset for B

Q: Calc weight changes when input **X**=[0,0,1] presented



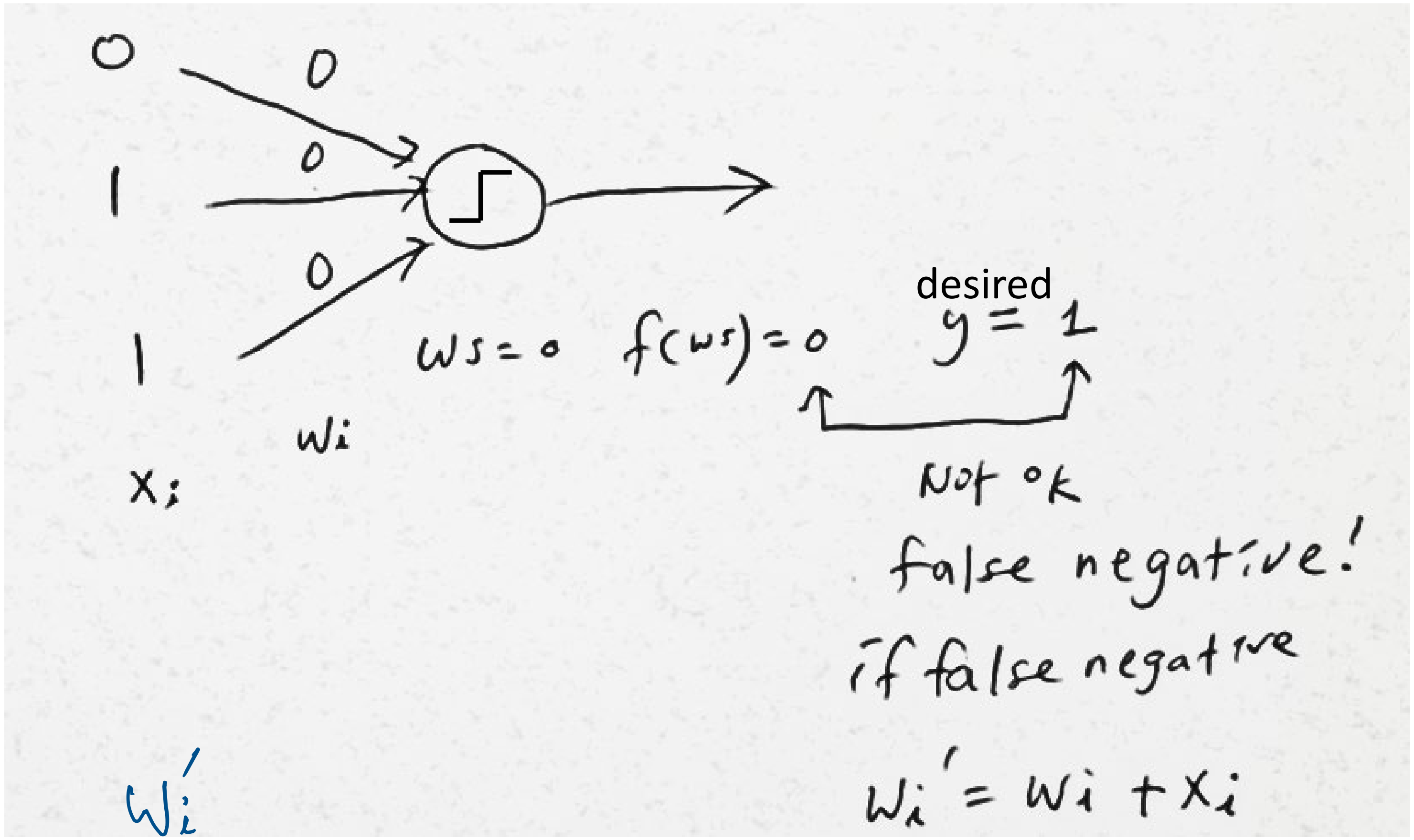
X_1	X_2	1	y
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

$$f(ws) = \begin{cases} 1 & \text{if ws} > 0 \\ 0 & otherwise \end{cases}$$

desired

Q: Weight changes when X=[0,1,1] presented? It becomes $\begin{bmatrix}0\\1\\1\end{bmatrix}$

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



$$\begin{array}{ccccc} W_i & & X_i & & W_i' \\ \begin{bmatrix}0\\0\\0\end{bmatrix} & + & \begin{bmatrix}0\\1\\1\end{bmatrix} & \Rightarrow & \begin{bmatrix}0\\1\\1\end{bmatrix}\end{array}$$