Universidad de Alcalá

Introducción al Deep Learning

Prof. Ignacio Olmeda
AI LAB

INTRODUCTION

- As mentioned, there are many aproches to build AI systems, among them classification trees, nearest neighbors, bayesian models and so on.

- The diversity of models and aproches make AI so interesting and enriching , ideas that happen in one area can be adapted to another one improving models.

- Among the many models in AI, probably the most successful and popular are those based on artificial neural networks which are commonly known as *Deep Learning* models.

- Deep Learning has been successfuly applied to many areas, the advancements in the last few years have been spectacularand, in some cases, DL-based systems have reached and even exceeded human performance.

- Deep Learning is also at the heart of recent impressive developments such as the prediction of –virtually- the complete human proteome[1], a landmark in biosciences.

[1]Tunyasuvunakool et al (2021), Nature, vol. 596, August 2021, https://www.nature.com/articles/s41586-021-03828-1.pdf

- Deep Learning is, by itself, an extremelly rich area: it would be imposible to provide a complete introduction to the subject, but it is posible to review the most commonly used models which are employed in the majority of applications.

- Among them, *feedforward networks*, *sequential networks*, *convolutional networks* and *generative networks* are probably the most widely used models, these will be the object of this course.

ARTIFICIAL NEURAL NETWORKS MODELS

- As we will explain later, *Deep Learning* is just another way to call *Artificial Neural Networks* (ANNs) models.

- ANNs are based on simulating not just the functioning of the animal brain but, in fact, also its structure.

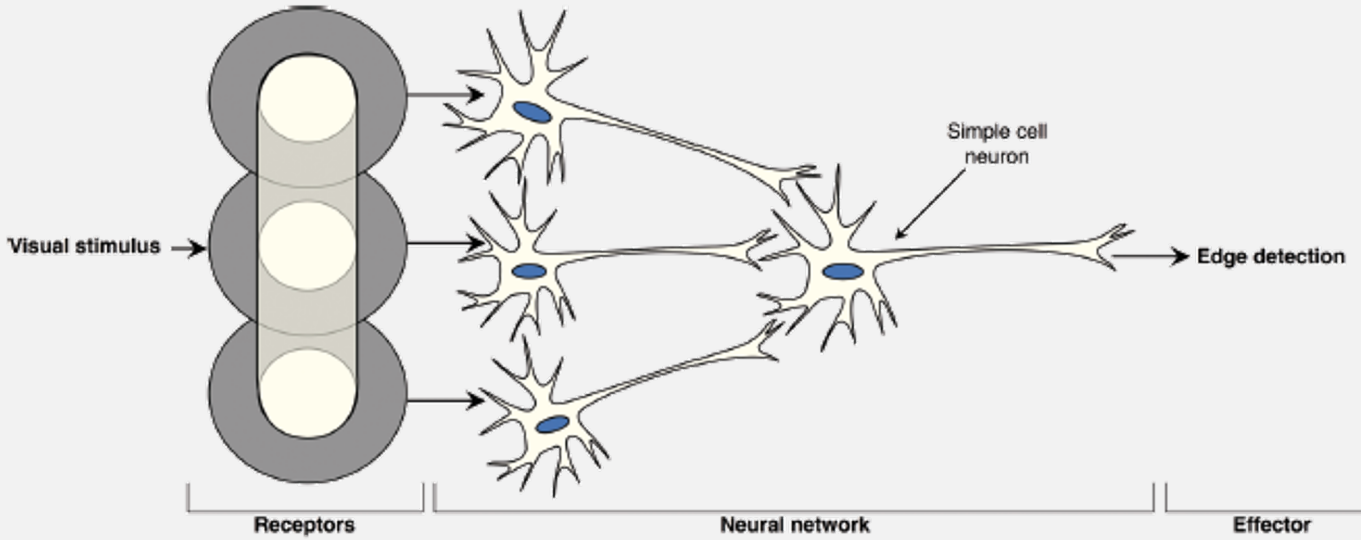
- Similarly to GAs, ANNs try to simulate the computational capabilities that emerge from biological systems; this strategy of "imitating" Nature has provided many of the most powerful solutions available in Engineering.

- The Human brain is probably the most complex mechanism known, but if we focus on the most simple properties and structures, still we will be able to create artificial systems enormously powerful.

- As known, in the animal brain, the basic information processing unit is the *neuron* conceptualized by Ramón y Cajal and Golgi at the begining of the XIX Century.

- Neurons exchange information through the *synapses*: electrochemical signals are propagated from the synaptic area through the *dendrites* towards the *soma*, the body of the cell.

- When a *threshold* is reached, the cell releases an activation signal through its *axon* towards neighboring neurons.



- Artificial Neural Networks (ANNs) are inspired by their physical analogues: similarly as with biological nets, the basic unit of an ANN is the *artificial neuron* or *node*.

- The nodes are highly simplified models that emulate the mechanism found in the biologic neurons.

- In what follows we will describe the most simple models which are, nowadays, the most widely used.

- Some models with a closer relation to actual neural networks (e.g. *Adaptive Resonace Theory* models, Grossberg 2020[2]) are out of the scope of this course but, in the future, they may provide even more powerful (and credible) models.

[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7330174/pdf/fnbot-14-00036.pdf

- The nodes are commonly arranged in *layers* that impose some hierarchical structure in ANNs: nodes of one layer receive, transform and transfer the weighted signals to other nodes in the subsequent layer.

- The *input layer* receives the information from outside, each of the nodes in this layer receive as inputs the values which represent the features of the problem to be solved.

- These inputs are transformed through an *activation function* (usually a simple nonlinear function) and passed through other nodes, previously multiplying them by a *weight* (any real number).

- In the following node, the inputs from the previous nodes are added and the process continues until an *output layer* is reached

- The output layer produces an output to the outside world which can be seen as the "reaction" of the network to the inputs received.

- If the network is properly built, the response to the inputs (the output of the network) should be the expected one, otherwise one would need to perform modifications until the actual output is equal to the expected one.

- This process of modificating the structure of the network until it perfroms correctly is what we call *learning*, and will be analyzed in detail later.

- As mentioned, the *input layer* and is in charge of receiving the external inputs, the last layer is called the *output layer* and has the function to transmit the result of the computation, for example, the forecast of some particular variable.

- The layers that lay between the input and output layers are called *hidden layers*, since they have no direct conection with the "enviroment".

- These kind of architectures are called *multilayer feedforward networks* or also -though not technically correct- *multilayer perceptrons*.

- As mentioned, ANNs models are analogues of Deep Learning.
- The term *deep learning* refers to the fact that this structure of stacking layers has some "depth".
- Before, approximately, 2010, these models were simply called ANNs models because due to computational constraints, only one or few hidden layers could be implemented.
- With the increase in computational power more hidden layers were added to ANNs models, networs were "deeper" and this explains the term *Deep Learning*.

PERCEPTRONS

- As mentiond, there are many alternative Deep Learning models: *Long Short Term Memory Networks*, *Hopfield Networks*, *Self Organizing Maps*, *Fuzzy Cognitive Maps*, *Generative Adverarial Networks*, *Convolutional Networks*, *Feed-forward Networks*, *Radial Basis Functions Networks*, *Bayesian Networks*, *Restricted Boltzmann Machines* and many others.

- Each of them have their own properties, structure (*topology*), learning algorithms and sometimes are better suited to particular problems.

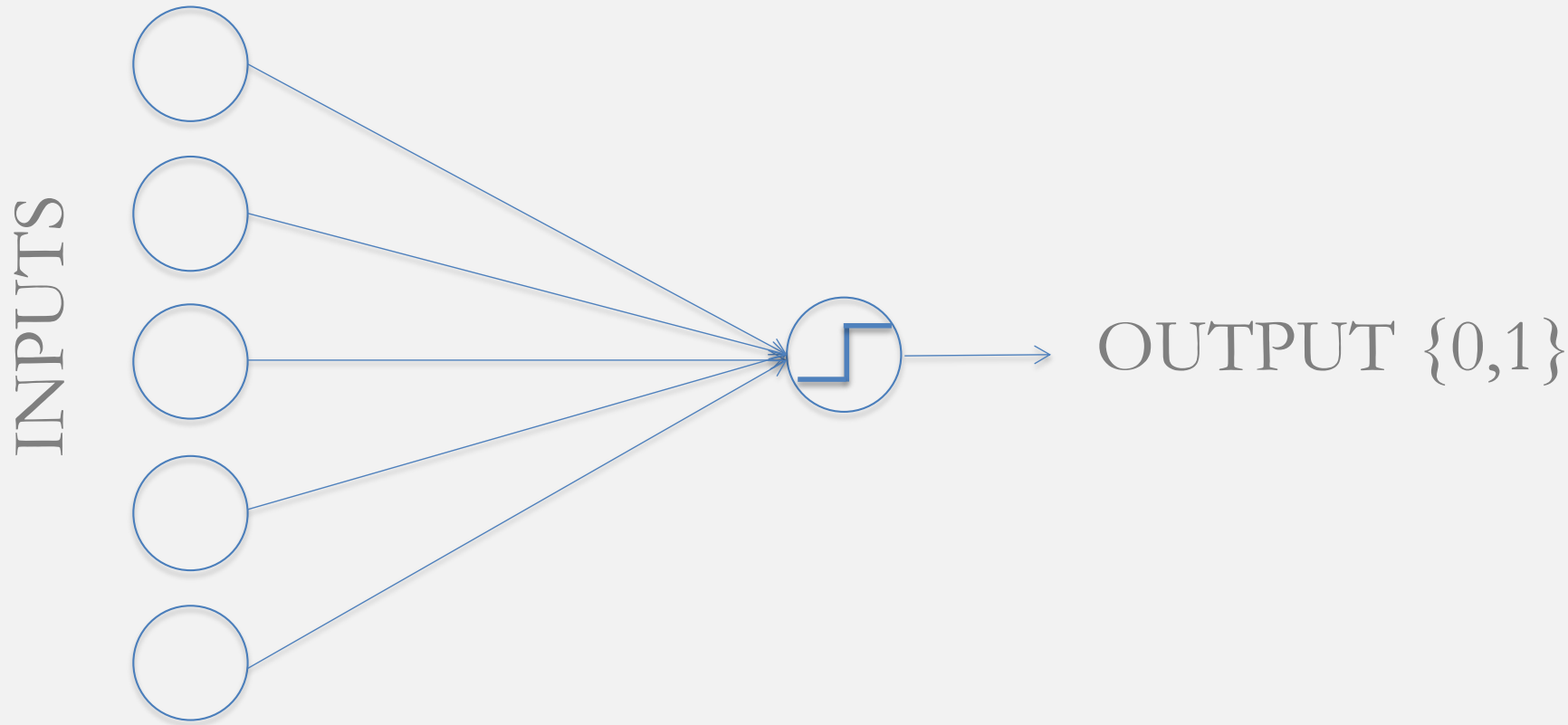
- Nevertheless, many concepts and techniques are shared and a good understanding of the most used models are enough to develop successful DL applications.

- The first architecture that we will analyze is called a *perceptron*.

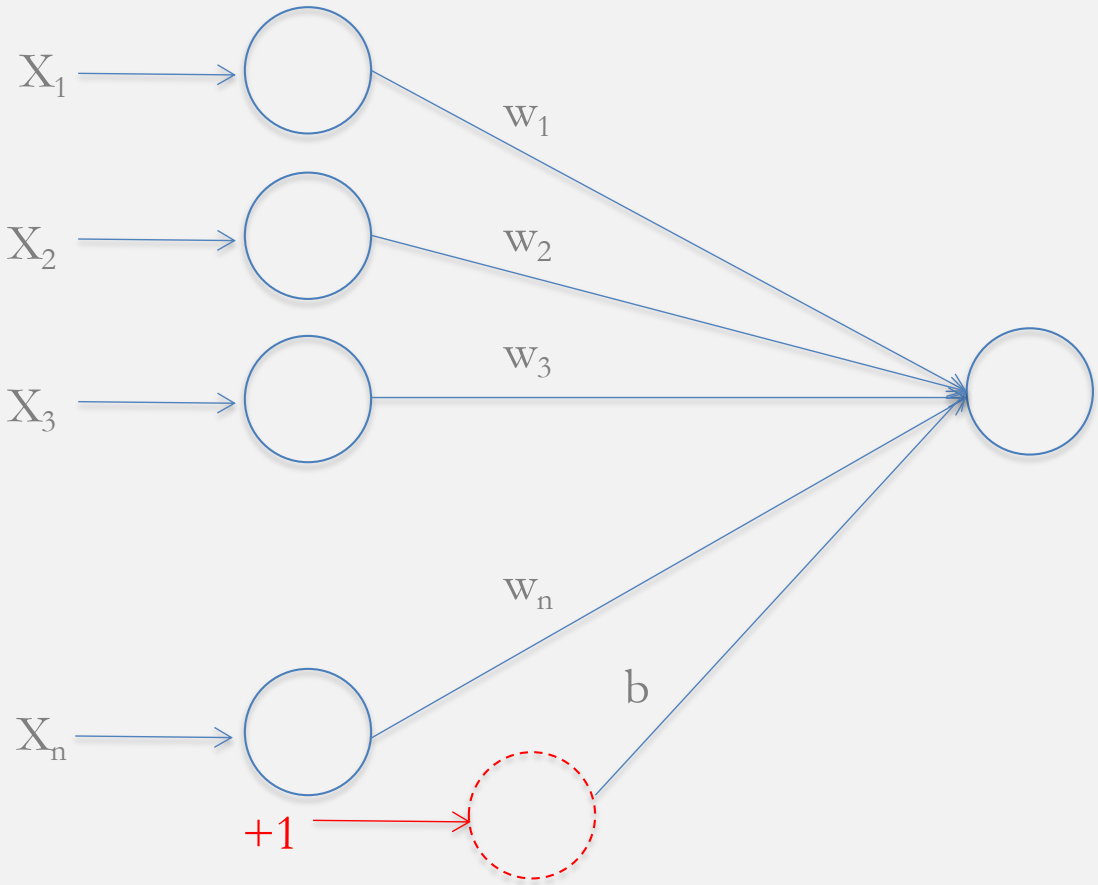
- Though the *perceptron* is too simple for most practical purposes, it allows to ilustrate several important aspects of DL models, particularly how neurons and neural networks work, some of their powers and limitations, and how *Learning* is performed.

- The *perceptron* it has its own historic value since it is one of the first ANNs models (Rosenblatt, 1958).

- A *perceptron* consists on a layer of input neurons that receive signals from the environment and an output layer, which produces a zero or a one, depending on whether a *threshold* is reached or not, schematically:



- A set of weighted links (*weights* $w_1,w_2,\ldots,w_n$) connects inputs x_i $i=1,2,..n$, to the output unit, i.e. the input by mutiplying it by the corresponding weight, $w_i x_i$.
- A special weight (noted as b in the figure), called the *bias*, is used and connects a hypothetic node whose value is always equal to one (+1) to the output unit, i.e.



- The inputs units just receive signals x_i from the environment and perform no operation while the output unit is a "firing" network which outputs 1 in case a threshold is reached and 0 otherwise.

$$\phi_{b,w}(x_1, x_2, ..., x_n) = \begin{cases} 1 \quad if \quad b + \sum_{i=1}^{n} w_i x_i > 0 \\ 0, \quad otherwise \end{cases}$$

- A perceptron could be used, e.g. to determine whether a particular image belongs to a class or not, for example whether it is the picture of a cat.

- In this case, the inputs could be e.g. black and white pixels of a 28 x 28 image (which gives 784 binary inputs, i.e. x_i, i = 1,2,…,784*);* an output of zero would mean that the picture is not a cat and one if it is a cat.

PERCEPTRONS´ LEARNING ALGORITHM

- The question is whether exists weights $w = (w_1, w_2, \ldots, w_n)$ so that the output of the perceptron $\Phi_{b,w}(x^k)$ equals the expected output $a^k \in \{0,1\}$.

- In such case, a further question is to determine how we can find such weights.

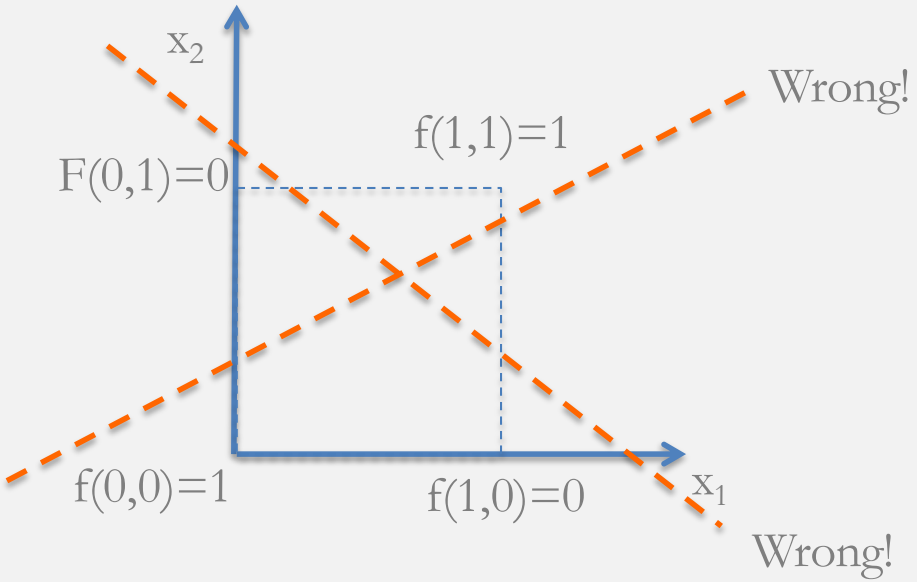
- Assume, for the moment, that the vector of weights $w_1, w_2, \ldots, w_n$ is random, obviously there are very few probabilities that, in our example, for a particular image $\boldsymbol{x}^k = \{x^k_1, x^k_2, \ldots, x^k_n\}$ the perceptron would produce the correct output a^k.

- The first question is not obvious because for some problem, even an apparently simple one, it could be impossible to find such weights.

- A paramount example is the XNOR function:

X_1	X_2	Y
1	1	1
0	0	1
1	0	0
0	1	0

- The XNOR is a trivial function $Y = f(X_1,X_2)$ with only four examples apparently easy to be learnt.
- Nevertheless it can be demonstrated (Minsky, 1969) that it does not exist a set of weights $w_1,w_2,\ldots, w_n, b$ so that they produce the correct output.
- The problem is that perceptrons just work for what are called *linearly separable functions*, i.e. problems for which the classes can be separated by a linear hyperplane, this is not the case of XNOR:



- The second question refers to how we can find such weights, in the case they exist.

- Let us see this, to simplify notation let $b = w_0$ and $x_0 = +1$.

- For linearly separable problems (for which such set of weights do exist), we can employ a very simple algorithm, called the *perceptron algorithm*.

- Note that learning is exactly this: finding the optimal weights that allows to perform a task under a performance criterion.

- The task is, in our case, correctly classifying the examples, the performance is the number of misclassifications and the perceptron algorithm is the learning algorithm.

Perceptron algorithm:

1. Set the weigths and bias equal to 0

$$w_0 = w_1 = w_2 \ldots = w_n = 0$$

2. For m iterations :

   If $\Phi_{b,w}(x^k) = a^k$ for every k then stop

   else, for every k, modify all weights by adding a term:

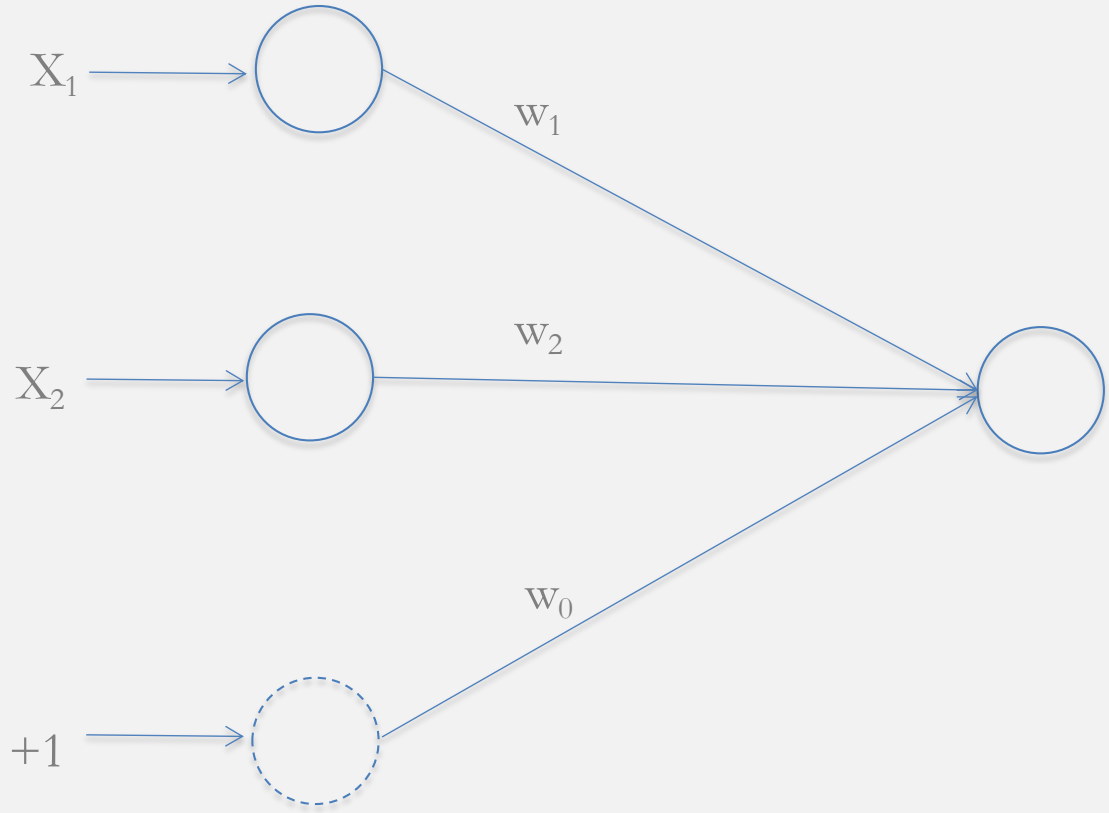
$$\Delta w_i = (a^k - \Phi_{b,w}(x^k))\, x^k{}_i$$

So that:

$$w'_i = w_i + \Delta w_i$$

- To understand how the perceptron algorithm works, let us assume that for some particular example k the correct class is $a^k=1$ while the output is $\Phi_{b,w}(\boldsymbol{x}^k)=0$, this means that $a^k - \Phi_{b,w}(x^k) = 1$.
- So we have $\Delta w_i = (a^k - \Phi_{b,w}(x^k))\, x^k{}_i = x^k{}_i$ which is always –not strictly- positive (pixels are either 0 or 1) so weights will increase or remain the same and the next time the output will be higher making the perceptron to be "less wrong".
- The perceptron algorithm is obviously iterated along different *epochs*.

- Example: Assume we have and we want to find the weights to implement the XNOR function



- Let us present the third pattern (randomly chosen):

X_1	X_2	Y
1	1	1
0	0	1
1	0	0
0	1	0

- We have:

$$\boldsymbol{x}^3 = (x^3_0, x^3_1, x^3_2) = (1,1,0)$$

note that we create an artificial input feature x^k_0 which always is +1, for all the examples k=1,2,3,4.

- According to the algorithm we initialize the weights to zero:

$$w_0 = w_1 = w_2 = 0$$

- The output for this example will be:

$$\Phi_w(\boldsymbol{x}^3) = \Phi(w_0 x_0 + w_1 x_1 + w_2 x_2) = \Phi(0 \text{ x } 1 + 0 \text{ x } 1 + 0 \text{ x } 0) = 0$$

- Since $a^3 = 0 = \Phi_w(\boldsymbol{x}^3)$ there is no change in weights:

$$\Delta w_i = (a^3 - \Phi_w(x^3))\, x^3{}_i$$

$$\Delta w_0 = (a^3 - \Phi_w(x^3))\, x^3{}_0 = (0\text{-}0)\ \mathrm{x}^{\mathrm{k}}{}_0 = 0 \text{ x } 1 = 0$$
$$\Delta w_1 = (a^3 - \Phi_w(x^3))\, x^3{}_1 = (0\text{-}0)\ \mathrm{x}^{\mathrm{k}}{}_1 = 0 \text{ x } 1 = 0$$
$$\Delta w_2 = (a^3 - \Phi_w(x^3))\, x^3{}_2 = (0\text{-}0)\ \mathrm{x}^{\mathrm{k}}{}_2 = 0 \text{ x } 0 = 0$$

- Now, let us present the second pattern (randomly chosen):

X_1	X_2	Y
1	1	1
0	0	1
1	0	0
0	1	0

- We have:

$$\boldsymbol{x}^2 = (x^3_0, x^3_1, x^3_2) = (1,0,0)$$

- The output for this example will be:

$$\Phi_w(\boldsymbol{x}^3) = \Phi(w_0 x_0 + w_1 x_1 + w_2 x_2) = \Phi(0 \text{ x } 1 + 0 \text{ x } 1 + 0 \text{ x } 0) = 0$$

- We have $a^3 = 1 \neq \Phi_w(\boldsymbol{x}^3) = 0$

$$\Delta w_i = (a^2 - \Phi_w(x^2))\, x^2_i$$

$$\Delta w_0 = (a^2 - \Phi_w(x^2))\, x^2_0 = (1\text{-}0)\, x^k_0 = 1 \text{ x } 1 = 1$$
$$\Delta w_1 = (a^2 - \Phi_w(x^2))\, x^2_1 = (1\text{-}0)\, x^k_1 = 1 \text{ x } 0 = 0$$
$$\Delta w_2 = (a^2 - \Phi_w(x^2))\, x^2_2 = (1\text{-}0)\, x^k_2 = 1 \text{ x } 0 = 0$$

- So we change the weights accordingly:

$$w'_0 = w_0 + \Delta w_0 = 0 + 1 = 1$$
$$w'_1 = w_1 + \Delta w_1 = 0 + 0 = 0$$
$$w'_2 = w_2 + \Delta w_2 = 0 + 0 = 0$$

- Notice that the first weight has changed so that now we have:

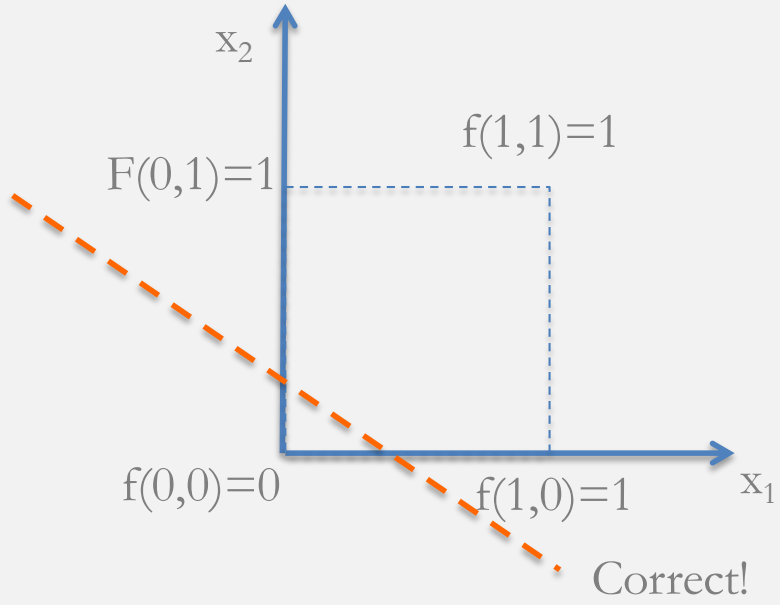
$$w_0 = 1$$
$$w_1 = w_2 = 0$$

- In general we would continue iterating the algorithm until it makes no error in any of the examples but note that, since XNOR is not lineraly separable, optimal weights can not be found, i.e., the algorithm would never *converge*.

- A different situation happens if the function is linearly separable, in such case, optimal weights do exist and the algorithm would converge.

- That is the case, e.g. of the OR function, which is linearly separable so that there exists a perceptron that can implement it:

X_1	X_2	Y
1	1	1
0	0	0
1	0	1
0	1	1



- We will see, using a program in Python, that it is very easy to find the weights so that the OR function can be implemented in a perceptron.

- As mentioned, the perceptron is too simple to be considered a useful model, but it has allowed to ilustrate very important points which are essential in general cases:
    - The need to be sure that the architecture is appropriate for the problem at hand
    - The need to have an efficient learning algorithm to find the optimal weights
- It also allowed us to ilustrate the iterative nature of learning algorithms and how errors are used to update the corresponding parameters of the model.

- Even though the problems considered (XNOR, OR) are very simple (few examples, totally deterministic, few and trivial features…), so that in practical problems things will be more complicated, still the same principles and similar techniques as the ones seen here will apply.