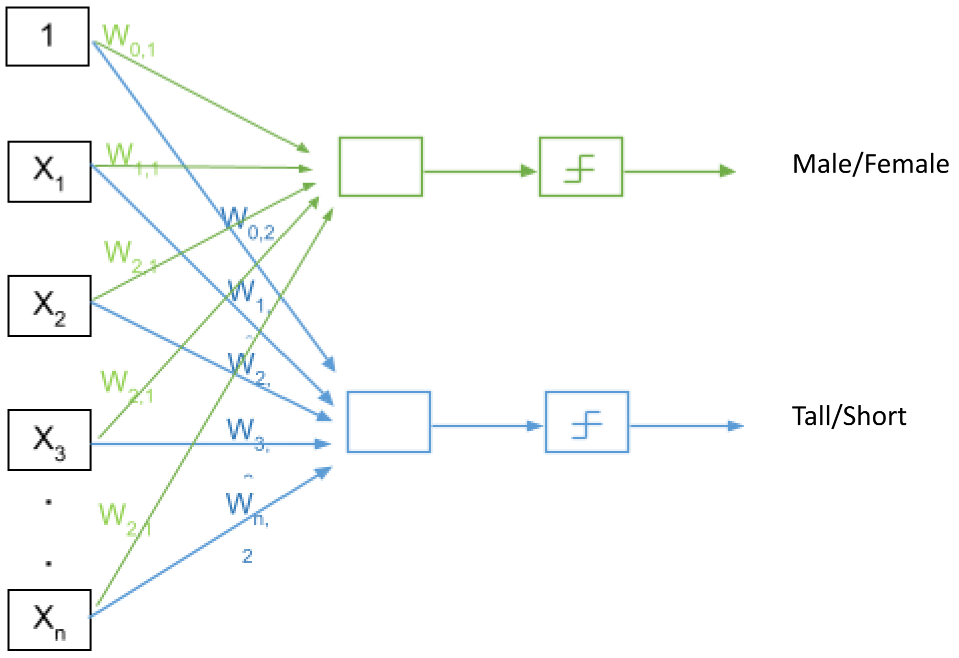
Lecture 7 – Introduction to Multi-Layer Neural Networks

Multi-Perceptron Single Layer Neural Networks

A primitive neural network can be constructed from multiple perceptrons arranged in a single layer. While these networks can learn to classify inputs into more than one category (e.g., Male/Female AND Tall/Short) all of the underlying algorithms covered above still apply. This is because the individual perceptrons are not interconnected in any way.

We can think of a single layer network as multiple independent perceptrons operating "in parallel".

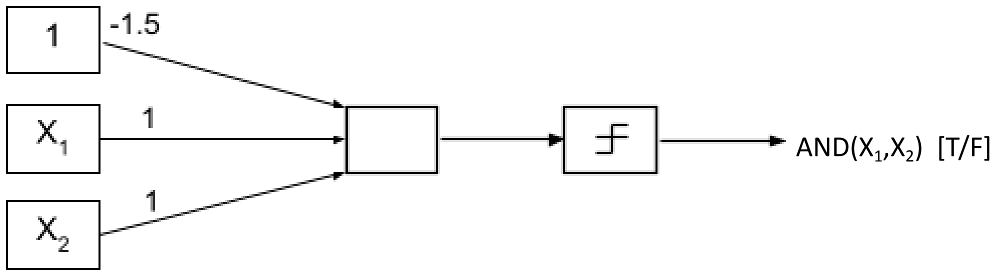
General Schematic



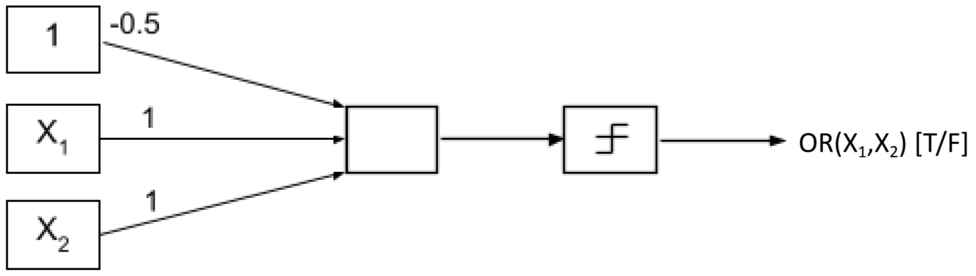
Basic Logic Operators (AND, OR, NOT) implemented as perceptrons

We made the claim some time ago that any computable function can be implemented as a (multi-layer) network of MCP neurons. We can lay the foundation for a proof that this statement is true by showing how the underlying AND, OR, and NOT logic operations can be implemented using MCP neurons.

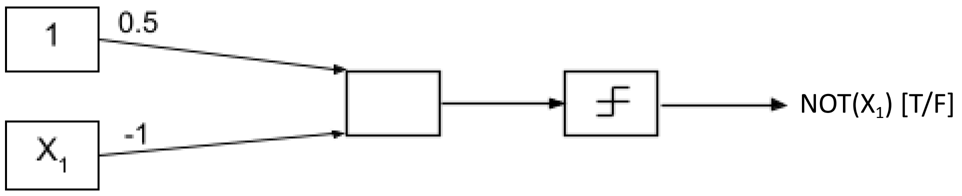
2-input AND implemented as a perceptron



2-input OR implemented as a perceptron



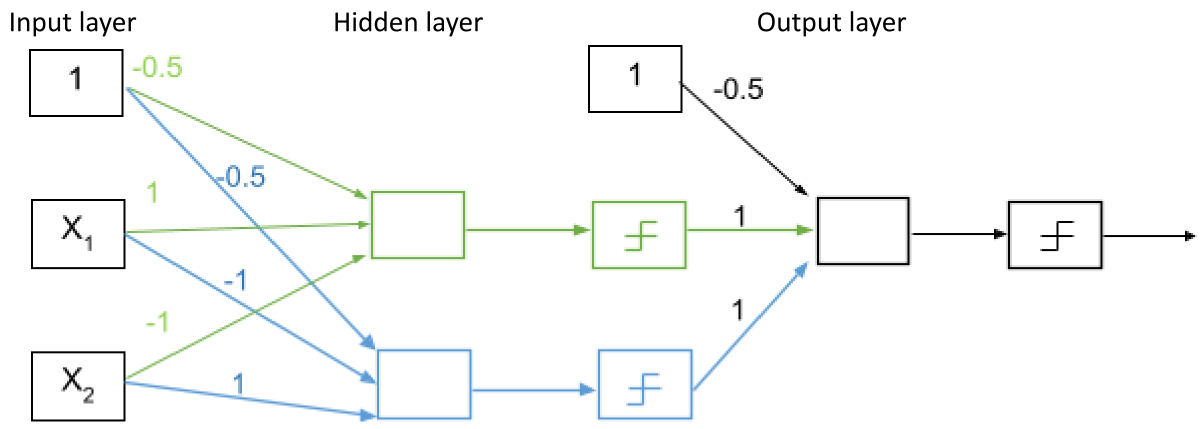
Single input NOT implemented as a perceptron



Now that we have the basic logic operations, we still need to show how these individual perceptrons can be “wired” together into a multi-level perceptron capable of solving arbitrary logic problems.

Multi-Layer Perceptron Neural Network

Multi-layer perceptron networks can solve the XOR problem. Consider the following two-level network with hand coded weights, where X_1 and X_2 are limited to the values 0 and 1.



A Hand-Coded Solution to the XOR Problem using a Multi-Layered Perceptron Network

We have a two layer network. In the first layer there are two perceptrons **GREEN** and **BLUE**. These two perceptrons feed into a second layer consisting of a single **BLACK** perceptron.

Again, assuming X_1 and X_2 are limited to the values 0 and 1.

- The **GREEN** node, defined by $\mathbf{-0.5 + X_1 - X_2 >= 0}$ can fire only when $X_1 = 1$ and $X_2 = 0$.

X_1	X_2	$\sum$	Output
0	0	-0.5	0
0	1	-1.5	0
1	0	**0.5**	1
1	1	-0.5	0

- The **BLUE** node, defined by $\mathbf{-0.5 - X_1 + X_2 >= 0}$ can fire only when $X_1 = 0$ and $X_2 = 1$.

X_1	X_2	$\sum$	Output
0	0	-0.5	0
0	1	**0.5**	1
1	0	-1.5	0
1	1	-0.5	0

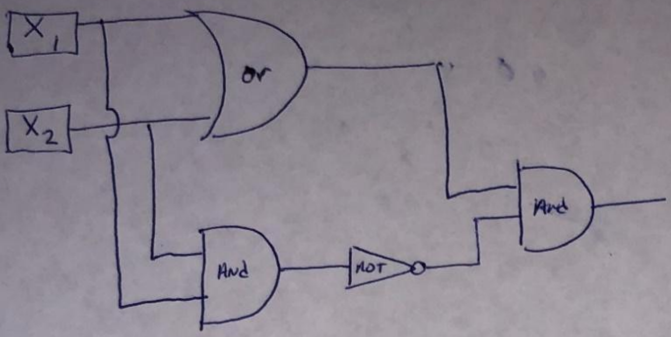
- The **BLACK** node, defined by **-0.5 + GREEN + BLUE >=0** can fire when **GREEN** = 1 or **BLUE** = 1, Note that by the structure of the network **GREEN** and **BLUE** cannot both be 1 at the same time.

GREEN	**BLUE**	$\sum$	Output
0	0	-0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	N/A	N/A

Implementing XOR using perceptrons (in a more standard way, directly from the logic gates)

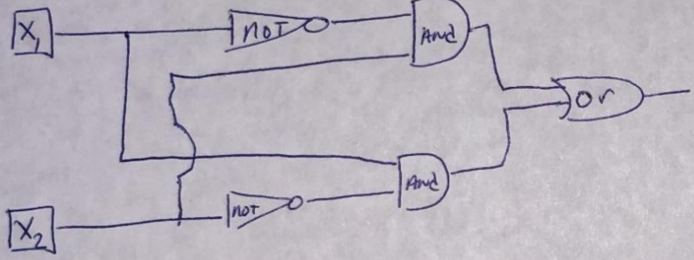
Here are two additional implementations of XORs.   The second one has been converted to a multi-layer MCP network.

XOR$_1$

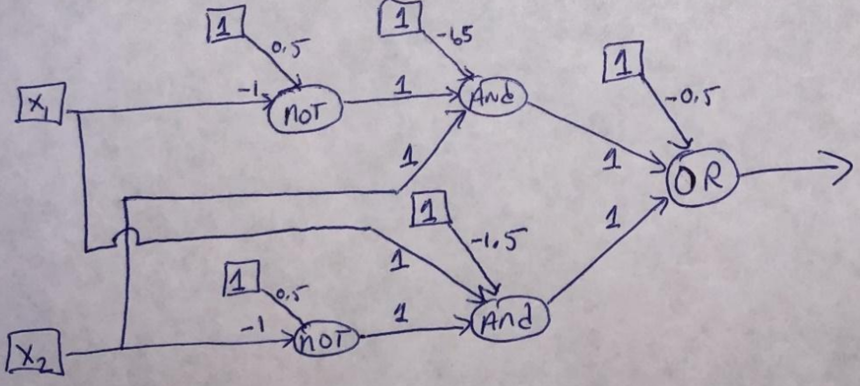


X1
X2
or
And
NOT
And

XOR$_2$



X1
NOT
And
or
And
NOT
X2



1
0,5
1
-1,5
1
-0,5
X1
-1
NOT
1
And
1
1
OR
1
1
-1,5
1
0,5
-1
NOT
1
And
X2

Moving Beyond Perceptrons

As stated previously, the fundamental problem with trying to build multi-layer perceptron networks is that the perceptron learning rule only works for a single layer of (non-interacting) perceptrons. It cannot train intermediate (or "hidden") layers.

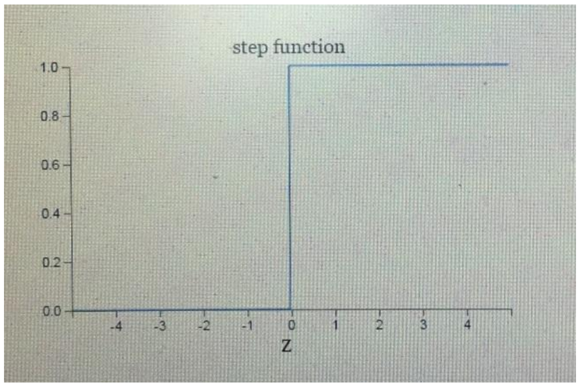
We need multiple layers of adaptive, non-linear hidden units. In other words, we need an efficient way of adapting all the weights, not just the final layer.

We will begin by introducing a neuron that is somewhat more advanced than a perceptron. Once we understand how this new neuron behaves we will look at how multi-layer networks of these neurons can be trained.

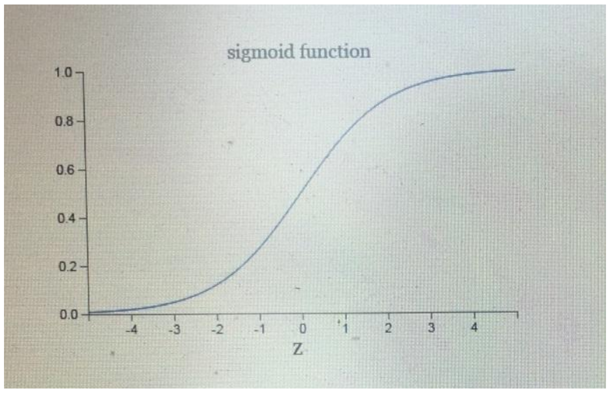
Sigmoid Neurons

Much of the material in this section is taken from the free online book [1] "Neural Networks and Deep Learning", Michael A. Nielsen, Determination Press, 2015. http://neuralnetworksanddeeplearning.com

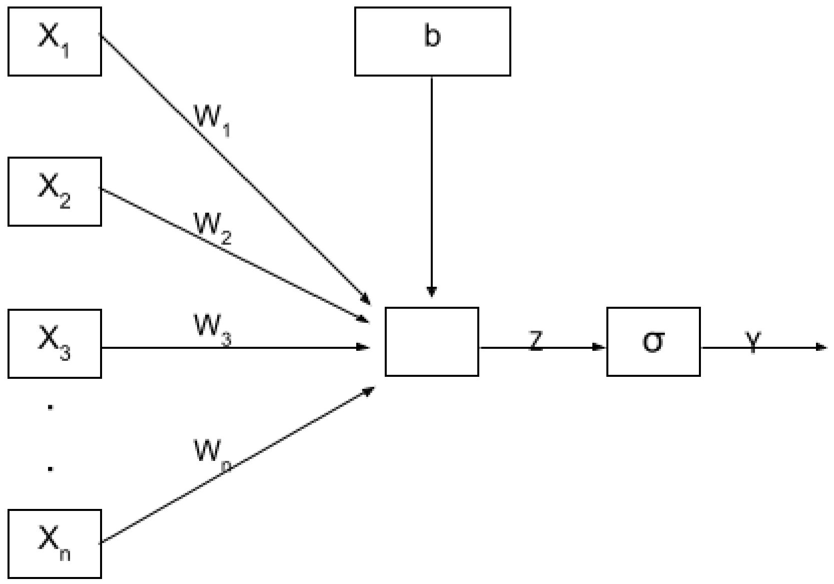
The output of a perceptron is either a 0 or 1. Crossing the threshold acts as a discontinuous step function.



A **sigmoid neuron** (also called as **logistic neuron**) is an artificial neuron with a real valued output between 0 and 1 inclusive, which we will call y. Whereas the output of a perceptron is a discontinuous step function, the output of a sigmoid neuron is a continuous function.



We can graphically represent a sigmoid neuron as follows:



You'll note that the sigmoid neuron looks almost identical to a perceptron. We still compute the weighted sum of the input values and then add in a bias **b**, we call the result of this computation **Z**. So, **Z = (∑ (i=1 to n) W_i X_i) + b** or equivalently the dot product of vector **W** with vector **X** plus the scalar bias **b** which can be written **W·X + b**. However, instead of sending this **Z** value to a threshold comparator and generating a 1 or 0, we provide **Z** as input into a sigmoid function **σ** [sigma] which generates the output of the sigmoidal neuron, **Y**.

The output of the sigmoid neuron, Y, can thus be defined mathematically as:

$$y = \boldsymbol{\sigma}(z)$$
$$\text{where } z = \left(\sum (i=1 \text{ to } n)\, W_i X_i\right) + b$$

or, equivalently, in matrix notation this looks like:

$$y = \sigma\left(\begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_{n-1} \end{bmatrix} \bullet \begin{bmatrix} x_0 & x_1 & x_2 & \vdots & x_{n-1} \end{bmatrix} + b\right)$$

W is 1 by n X is n by 1

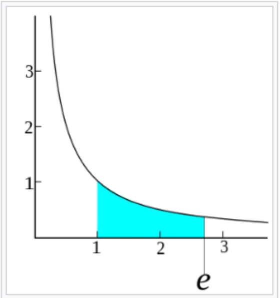
To make computer scientists happy I started my indices at 0 instead of 1. Note that W is a 1 by n matrix (vector) and X is an n by 1 matrix (vector), so their dot product is a scalar (single numerical value).

So, I've explained what Z is. But, what is **σ**?

The sigmoidal function **σ**(z) is defined as $\dfrac{1}{1+e^{-z}}$

What is **e** you ask? It is the Euler number ≈ 2.71828. While not that familiar to computer scientists, this number appears frequently in the natural sciences. It is the base of the natural logarithm and can be computed by the infinite series

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots$$



The graph is $y = 1/x$, and e is the unique number that makes the shaded area equal to 1.

Putting this all together the output of a sigmoidal neuron, y, is:

$$y = \frac{1}{1 + e^{-(W^{*}X+b)}}$$

Or, in a more "computer science-like" notation:

y = 1 / (1+exp(-(∑ (i=1 to n) W_i X_i) - b))

At first glance, the output function of a sigmoid neuron might look scary, but really it is a lot like a perceptron neuron except that the output is a continuous function rather than a discontinuous step function.

When the weighted sum of the inputs (plus bias), **Z**, is a large positive number then $\mathbf{e^{-z} \approx 0}$

$$e^{-z} = \frac{1}{e^{z}} = \frac{1}{2.718^{LargePositve}} = \frac{1}{VeryLargePositive} \approx 0$$

$$y = \frac{1}{1+e^{-z}} \approx \frac{1}{1+0} = 1$$

And the output of the sigmoidal neuron $\mathbf{y \approx 1}$ which is similar to the output of a perceptron under similar conditions.

When the weighted sum of the inputs (plus bias), **Z**, is a large negative number then $\mathbf{e^{-z} \approx \infty}$

$$e^{-z} = e^{-LargeNegative} = e^{LargePositive} = 2.718^{LargePostive} = VeryLargePositive$$

$$y = \frac{1}{1+e^{-z}} \approx \frac{1}{1+VeryLargePositive} \approx \frac{1}{VeryLargePositive} \approx 0$$

And the output of the sigmoidal neuron $\mathbf{y \approx 0}$ which is also similar to the output of a perceptron under similar conditions.

The primary difference is, again, that $\boldsymbol{\sigma}(z)$ is a continuous function that is thus differentiable. Small changes in the weights ($\boldsymbol{\Delta}\ \mathbf{w_j}$) and in the bias ($\boldsymbol{\Delta}\ \mathbf{b}$) will produce a small change in the output (**Δ output**) of the neuron.