2

MCCULLOCH–PITTS NEURON

Vasant Honavar

Artificial Intelligence Research Group
Department of Computer Science
Iowa State University
Ames, Iowa 50011

1 MCCULLOCH–PITTS NEURON

We start by introducing an extremely simplified model of a biological neuron, based on the early work of McCulloch and Pitts. This model is variously refered to as McCulloch-Pitts Neuron, Threshold Neuron, Threshold Logic Unit (TLU) and perceptron.

What follows is a caricature that describes the working of nervous systems: A nervous system is an organized network of neurons. A typical neuron consists of three parts: the dendrites, the cell body, and the axon (also called the nerve fiber). The dendrites carry nerve signals toward the cell body, while the axon carries the signal away from the cell body. Neural circuits are formed from groups of neurons arranged with the end branches of the axon lying close to the dendrites of another neuron. In the human brain, each neuron can interact with thousands of other neurons. The point of contact between the components of two neurons is called a synapse. A small microscopic gap between the two neurons exists at a synapse. It is known that the ease of neural signal transmission across the synapse is altered by activity in the nervous system — a possible mechanism for learning. When a neuron receives input from other neurons, the electro-chemical processes involved cause its voltage to increase. When the voltage exceeds a certain *threshold*, it results in a volley of nerve

impulses that travel down the axon (thereby stimulating other neurons and so on). Or, the neuron is said to *fire*.

In McCulloch-Pitts model, numerical parameters called *weights* $w_1, \cdots, w_n$ model the strength of synaptic coupling between neurons. $x_1, \cdots, x_n$ model the inputs. T models the threshold. For mathematical convenience, T is replaced by a weight $-w_0$ and a fictitious input x_0 which is always constant (typically equal to unity) is introduced.

$y = 1 \; if \; \sum_{i=1}^{n} w_i x_i > T$
$or \sum_{i=1}^{n} w_i x_i - T > 0$
$or \sum_{i=0}^{n} w_i x_i > 0$
$or \mathbf{W} \cdot \mathbf{X} > 0; and$
$y = 0 \; otherwise$

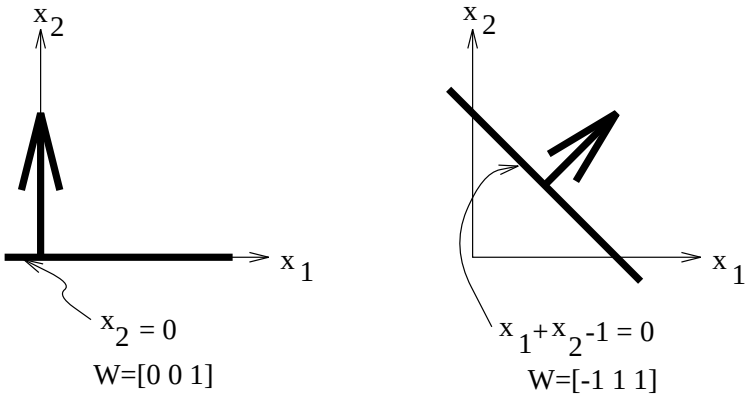

Figure 1 McCulloch-Pitts Neuron. $\mathbf{W} = [w_0, \cdots, w_n]^t$ stands for the weight vector; $\mathbf{X} = [x_0, \cdots, x_n]^t$ stands for the input vector; where $\mathbf{A}^t$ stands for the transpose of vector $\mathbf{A}$

2 COMPUTATIONAL CAPABILITIES OF A TLU

2.1 Connection with boolean logic

Suppose $x_i \in \{0, 1\}$ (i.e., the inputs are binary), and $x_0 = 1$. (A threshold neuron with binary inputs is called a binary threshold neuron). Suppose $w_1 = w_2 = \cdots = w_n = 1$. Consider a special case with $n = 2$. As evident from figure 3, this neuron functions as a 2-input AND gate.

x_1	x_2	$\mathbf{W} \cdot \mathbf{X}$	y
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1

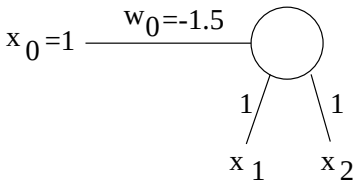

Figure 2 A threshold neuron that functions like an AND gate

Similarly, it is possible to select weight values so that an n-input threshold neuron functions as an n-input AND gate or as an n-input OR gate. It is also straightforward to select weights so that a 1-input threshold neuron (i.e., $n = 1$) functions as a NOT gate. Since any boolean function can be expressed in terms of AND, OR, and NOT functions, we have the following theorem:

Theorem:
A sufficiently large network of binary threshold neurons can compute any arbitrary boolean function.

Proof: Omitted.

McCulloch and Pitts also went on to prove that arbitrary finite state automata can be realized using binary threshold logic units with unit time delay (between input and output).

Since it is known from the theory of computation (developed by Turing, Kleene, Church, Rosser, Post, and others) that any arbitrary algorithm can be realized by a sufficiently large finite state machine, we can assert that there exist networks of threshold neurons that can compute any computable function.

Although any boolean function can be computed by an appropriately designed network of binary threshold neurons, there are functions that cannot be computed by a *single* binary threshold neuron. Exclusive OR (EXOR) is one such function. To see this, consider the constraints on weights that need to hold in order for a neuron to compute a 2-input EXOR:

$$
\begin{array}{ccc}
x_1 & x_2 & y \\
0 & 0 & 0 \rightarrow w_0 \leq 0 \\
0 & 1 & 1 \rightarrow w_0 + w_2 > 0 \\
1 & 0 & 1 \rightarrow w_0 + w_1 > 0 \\
1 & 1 & 0 \rightarrow w_0 + w_1 + w_2 \leq 0
\end{array}
$$

Clearly, there is no set of weights that simultaneously satisfy each of these 4 inequalities. Thus we have the following theorem:

Theorem: There exist logical functions that cannot be realized by a single binary threshold neuron.

Definition: The class of boolean functions that can be realized by a single binary threshold neuron are called boolean threshold functions.

In the light of the above, it is natural to ask what fraction of the number of possible n-input boolean functions can be realized by an n-input binary threshold neuron. To this day, we do not have an exact closed-form formula that provides an answer to this question. However, the number of possible threshold functions have been enumerated for $n \leq 6$. Let us consider the case for $n = 2$. There are 2^{2^2} boolean functions of 2 inputs. (There are 4 binary patterns:00, 01, 10, and 11. A boolean function partitions the set of patterns into 2 classes and there are $\left(\begin{array}{c} 4 \\ 0 \end{array}\right) + \left(\begin{array}{c} 4 \\ 1 \end{array}\right) + \left(\begin{array}{c} 4 \\ 2 \end{array}\right) + \left(\begin{array}{c} 4 \\ 3 \end{array}\right) + \left(\begin{array}{c} 4 \\ 4 \end{array}\right) = 16$ ways to do this). Of these, 14 can be shown to be threshold functions. Before you jump to the conclusion that the number $N_T(n)$ of n-input boolean threshold functions is a relatively large fraction of the number $N_B(n)$ of n-input boolean functions, it must be noted that as n increases, threshold functions form a vanishingly small subset of possible boolean functions. In fact, it can be shown that $N_T(n) \leq 2^{n^2}$. Recall that $N_B(n) = 2^{2^n}$. It follows that the limit of $\frac{N_T(n)}{N_B(n)}$ approaches 0 as n approaches infinity.

2.2 Connection with Geometry

Consider 2-input boolean threshold neurons. The input patterns can be thought of as points in 2-d Euclidian space.

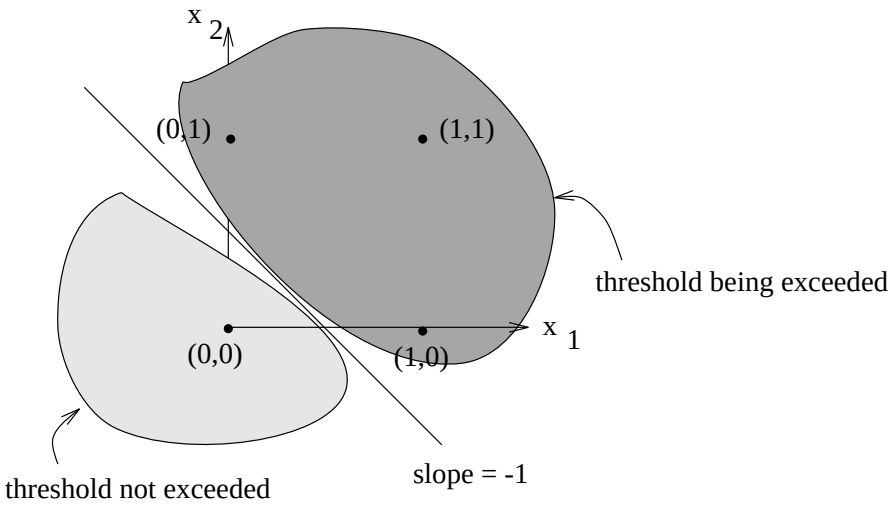

Figure 3

Suppose we draw a line given by $x_2 = -x_1 + 0.5$ in this space. The equation of the line can be written as $w_1 \cdot x_1 + w_2 \cdot x_2 + w_0 = 0$ where $w_1 = 1, w_2 = 1, w_0 = -0.5$. Thus, a 2-input threshold neuron partitions a 2-dimensional Euclidian space into two regions (of half-spaces). One one side of the line are points for which the neuron produces an output of 1. On the other side are points for which the neuron produces an output of 0. We have used the convention that if a point falls on the line, its output is 0. (But other variations on this leave the computational model essentially unchanged).

Can this be generalized into n-dimensional space?

As shown in the figure, a 3-input threshold neuron realizes a 2-D plane that partitions a 3D Euclidean space into two half-spaces. We can generalize this example to threshold neurons with an arbitrary, but finite number of inputs (n). In general, an n-input threshold neuron with weight vector $\mathbf{W} = [w_0 \cdots w_n]^t$ implements an $n-1$ dimensional hyperplane that divides the n-dimensional Euclidean space into two half-spaces. The equation of the plane is given by: $\mathbf{W} \cdot \mathbf{X} = 0$ (for some choice of $\mathbf{W}$), $\mathbf{X} = [1, x_1 \cdots, x_n]^t$ defines the axes of the n-dimensional space. The values of the components of $\mathbf{W}$ define the location

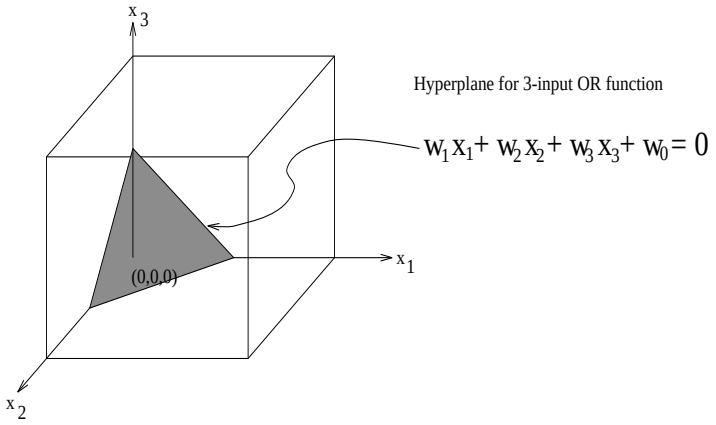

Figure 4 A 3-input threshold neuron implements a 2-dimensional plane

and orientation of the hyperplane. The hyperplane has two sides: a positive (+ve) side and a negative (-ve) side. A point $\mathbf{X}_k = [x_{0k} \cdots x_{nk}]$ is said to fall on the +ve side of $\mathbf{W} \cdot \mathbf{X} = 0$ if $W \cdot \mathbf{X}_k > 0$ and the -ve side if $\mathbf{W} \cdot \mathbf{X}_k < 0$.

Example: $\mathbf{W} = [w_0 \cdots w_3]^t = [1\ 1\ -1\ 1]^t$ and $\mathbf{X}_1 = [1\ 0\ 1\ 0]^t$
Question: What side of the hyperplane defined by $\mathbf{W} \cdot \mathbf{X} = 0$ does $\mathbf{X}_1$ fall?
$\Rightarrow$ On the plane since $\mathbf{W} \cdot \mathbf{X}_1 = 0$.

Note that although our examples have used binary inputs, in general, threshold neurons can have multi-valued inputs or real-valued inputs.

A Little Geometry

Because of the close relation between n-input threshold neurons and n-dimensional Euclidean space, we will digress to review some geometry.

- The equation of $n-1$ dimensional hyperplane in n-dimensional space is given by
$$w_0 + w_1 x_1 + \cdots + w_n x_n = 0 \tag{2.1}$$
- The orientation of the plane is governed by $(w_1 \cdots w_n)$ and its location by its distance form the origin is governed by w_0.
- The magnitude of the perpendicular vector distance of the hyperplane (1.1) from the origin is
$$\frac{|w_0|}{\sqrt{w_1^2 + \cdots + w_n^2}}$$

- The distance of an arbitrary point given by $(x_1 \cdots x_n)$ from the plane (1.1) is

$$\frac{|w_1 x_1 + \cdots + w_n x_n + w_0|}{\sqrt{w_1^2 + \cdots + w_n^2}}$$

Examples:

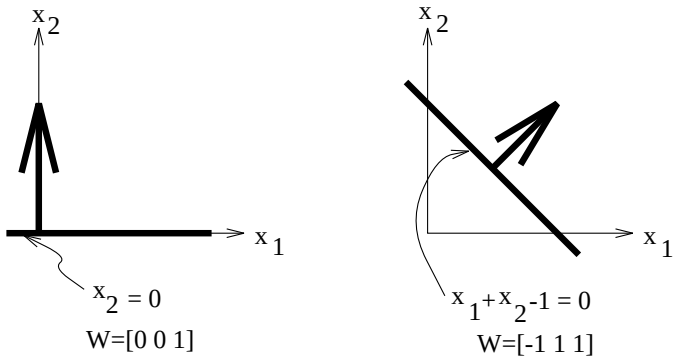

Figure 5 Hyperplanes and Direction of Unit Normal Vectors

3 SUMMARY

In summary, a threshold neuron is a highly simplfied computational model of a neuron. As we have seen, even such a simplified model has fairly interesting computational properties. In particular, a threshold neuron, given an appropriate choice of the weights, is capable of computing any threshold function. This raises the question as to whether such a device can be *taught* a desired (but a-priori unknown) threshold function from examples of the desired inputs and the corresponding outputs. If this could be done, as Rosenblatt and other early researchers in neural computation realized, threshold neurons could be used to build trainable pattern classifiers. Such pattern classifiers can potentially be used to solve pattern recognition problems that arise in diagnosis, handwriting recognition, etc. Clearly, since the exact function computed by a threshold neuron depends on the choice of weights, an obvious possibility for learning is to modify the weights so as to correctly classify a given training set. If the training set is *representative* of the domain of interest, one can hope that the resulting classifier would perform quite reasonably on novel input patterns. More on this later. We will examine some algorithms for learning threshold functions in the next chapter.