Chapter 11: Perceptrons And Logic Gates

11.1 INTRODUCTION

The translation of neural function into the operations of a two-valued logic was a critical step in the development of artificial neural networks, because it permitted McCulloch and Pitts to develop proofs about the potential power of their models (McCulloch & Pitts, 1943). However, one limitation of the McCulloch-Pitts architecture was that their networks did not learn. Is it possible to use a general learning rule and, using a set of stimulus-response examples, teach a network how to compute basic logical operations? The exercises in this chapter begin to explore this possibility.

11.2 BOOLEAN ALGEBRA

11.2.1 THE WORK OF BOOLE

In his obituary for George Boole (1815-1864), the Reverend Robert Hartley wrote "the facts of his personal history are few and simple, but they serve to illustrate how a man of humble origin, with very slender aids from without, may, by the force of genius and the labour of research, rise to a position of great eminence" (Boole, 1952) p. 426). Boole, who was self taught, made enormous contributions to the fields of logic and probability. Interestingly, for a period that began not long after his death, Boole's work was largely forgotten. However, early in the 20th century it reemerged, in a modified format, as a central component in the development of modern computers.

Boole's major work was his monograph entitled *An Investigation Of The Laws Of Thought, On Which Are Founded The Mathematical Theories Of Logic And Probabilities* (Boole, 1854). Boole's pioneering achievement in this book was the translation of basic logical operations into the operations of common algebra, with the goal of using mathematical operations to advance logical theory. He wrote that his intent was "to investigate the fundamental laws of those operations of the mind by which reasoning is performed; to give expression to them in the symbolical language of a Calculus, and upon this foundation to establish the science of Logic and construct its method; to make that method itself the basis of a general method for the application of the mathematical doctrine of Probabilities; and, finally, to collect from the various elements of truth brought to view in the course of these inquiries some probable intimations concerning the nature and constitution of the human mind" (p. 1).

In order to use mental operations to inspire his formal account, Boole (1854) took language as his starting point. "That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted" (p. 26). Boole observed that the fundamental elements of language are signs or symbols. From this he went on to hypothesize that all the operations of language could be formalized from three basic components. The first were literal symbols (***X***, ***Y***, etc.) that represented things (or sets of things). The second were signs (+, -, X) representing operations by which new concepts of things could be created by combining existing symbols for things. The third was the sign of identity (=), which foreshadowed Boole's mathematical goal (i.e., the development of expressions that equated some strings of symbols with others).

After proposing these building blocks, Boole (1854) developed basic algebraic laws for relating them. Imagine that the symbol ***X*** represents some set of objects, and that the symbol ***Y*** represents a different set of objects. Boole's basic law was $X^2 = X$, which indicates that Boole used multiplication to define the intersection of sets. Using Boole's own example, if ***X*** represents "all objects that are white", and if ***Y*** represents "all objects that are sheep", then ***XY*** represents "all objects that are white sheep".

Why did Boole (1854) choose multiplication to represent intersection? "His system is completely interpretable as an algebra in which the 'elective symbols' are restricted to the numbers 0 and 1" (Lewis & Langford, 1959) p. 13). Boole's "fundamental law of thought" $\boldsymbol{X^2 = X}$ (Boole, 1854, p. 54) is only algebraically true if one substitutes either the number 0 or the number 1 for the symbol $\boldsymbol{X}$. "Thus it is a consequence of the fact that the fundamental equation of thought is of the second degree, that we perform the operation of analysis and classification, by division into pairs of opposites, or as it is technically said, by dichotomy" (p. 55). This dichotomous view of mental operations would later prove central to the definition of digital circuits

With dichotomy being a natural consequence of his fundamental law of thought, Boole (1854) went on to provide precise definitions of two special symbols, $\boldsymbol{0}$ and $\boldsymbol{1}$. In modern terms, $\boldsymbol{0}$ is equivalent to the empty set, and $\boldsymbol{1}$ is equivalent to the universal set (i.e., the set of all possible sets). With this modern context in mind, it is clear why Boole wrote that "whatever class of objects is represented by the symbol $\boldsymbol{X}$, the contrary class will be represented by $\boldsymbol{1 - X}$" (p. 53). From this Boole's definition of what we call the null set was possible: $\boldsymbol{X - X^2 = 0}$, or more frequently $\boldsymbol{X(1 - X) = 0}$.

In these definitions of the symbols $\boldsymbol{0}$ and $\boldsymbol{1}$ Boole (1854) uses algebraic operators other than multiplication. Boole used the algebraic sign "+" to represent the aggregation of sets, which was intended to be equivalent in meaning to the word "or". Boole used the algebraic sign "-" to represent the opposite of aggregation, which he termed exception. So, to represent the phrase "all sheep except those that are white" with our two example sets above, we would use expression $\boldsymbol{Y - XY}$.

What was Boole's (1854) goal in creating a notation in which linguistic or logical expressions could be acted upon algebraically? What Boole desired was a system in which one could start with some given information, and then derive valid new information by applying algebraic operators. In this system, the given information would be expressed as equations involving symbols representing sets. "We may in fact lay aside the logical interpretation of the symbols in the given equation; convert them into quantitative symbols, susceptible only of the values 0 and 1; perform upon them as such all of the requisite processes of solution; and finally restore to them their logical interpretation" (p. 76). In short, Boole proposed an approach in which logical expressions would be converted into numbers, new numerical statements would be derived via algebra, and then new, valid logical expressions would appear when these resultant numerical statements were converted back into logical expressions.

Boole's (1854) work was pioneering because it represented "for the first time that a complete and workable calculus is achieved, and that operations of the mathematical type are systematically and successfully applied to logic" (Lewis & Langford, 1959), p. 9). Boole's formalism was the basis for future developments of symbolic logic. "It is not to be denied that Boole's system is consistent and perfect within itself. It is, perhaps, one of the most marvelous and admirable pieces of reasoning ever put together" (Jevons, 1971) p. 67).

11.2.2 Modern Boolean Algebra And Shannon's Work

While Boole's (1854) algebra was pioneering and revolutionary, one of the reasons that it inspired future developments in symbolic logic was because Boole's system was cumbersome, idiosyncratic, and in some instances mysterious. "The quasi-mathematical methods of Dr. Boole especially are so mystical and abstruse, that they appear to pass beond the comprehension and criticism of most other writers, and are calmly ignored" (Jevons, 1971) p. 84).

For example, consider the aggregation of terms in Boole's (1854) system. There are some expressions of aggregation that are likely to be seen as algebraic expressions, but which are not logically interpretable. One example of such an expression is $\boldsymbol{1 + X}$, which Boole (p. 55) deemed uninterpretable "because we cannot conceive of the addition of any class $\boldsymbol{X}$ to the universe $\boldsymbol{1}$." The expression $\boldsymbol{X + Y}$ only yields a logical interpretation in the event that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are

disjoint sets, that is – in Boole's notation – only when $\boldsymbol{XY} = 0$. If $\boldsymbol{X}$ and $\boldsymbol{Y}$ share some elements, then the expression makes algebraic sense, but has no corresponding logical interpretation. In other words, if one were to translate a Boolean expression of aggregation $\boldsymbol{X} + \boldsymbol{Y}$ into common language, it would read $\boldsymbol{X}$ or $\boldsymbol{Y}$ but not both. Because of Boole's interpretation of "or", an expression like $\boldsymbol{X} + \boldsymbol{X}$ in algebra would have no corresponding logical interpretation.

Modifications of Boole's (1854) approach were intended to tighten the relationship between algebraic expressions and logical interpretation in such a way that every algebraic expression could be assigned an interpretation. One of the major modifications that was adopted by nearly all subsequent researchers was a broadening of the common interpretation of "or" to mean "either one, or the other, or both". "Thus what is the same as $\boldsymbol{A}$ or $\boldsymbol{A}$ is the same as $\boldsymbol{A}$, a self-evident truth" (Jevons, 1971) p. 25). This lead to another fundamental law, which Jevons called the law of unity: $\boldsymbol{X} + \boldsymbol{X} = \boldsymbol{X}$. The law of unity was uninterpretable in Boole's original system, but lead to several advantages in later symbolic logics. Other modifications included the elimination of the operations of subtraction and division, as well as the inclusion of a new relation $\boldsymbol{X} \subset \boldsymbol{Y}$ to indicate that one set was a proper subset of another. The modern version of Boolean algebra is represented in a formalization worked out by Ernst Schröder in 1890 (Lewis & Langford, 1959).

Boolean algebra was originally developed to be applied to sets of objects. However, it is equally applicable to assertable statements (Lewis & Langford, 1959). For instance, let $\boldsymbol{A}$ be the set of instances in which some statement *a* can be said to be true, and let $\boldsymbol{B}$ be the set of instances in which some statement *b* can be said to be true. The algebraic expression $\boldsymbol{AB}$ would therefore represent the set of instances in which both *a* and *b* are true. Similarly, the algebraic expression $\boldsymbol{A} \subset \boldsymbol{B}$ would represent the subset of instances that could be stated as "if *a* is true then *b* is true".

This interpretation of Boolean algebra can be further restricted so that it applies specifically to propositions (Lewis & Langford, 1959). A proposition is an assertable statement that can only be true or false. To make this restriction explicit in Boolean algebra, one must add the assumption that "for every element *a*, either $a = 0$ or $a = 1$" (p. 79). This assumption is a very particular interpretation of Boole's (1854) emphasis on the dichotomous nature of thought, and leads to a modern version of Boolean algebra that is called the two-valued algebra. As we have already seen, the two-valued algebra was used as the formalism to provide a logical description of neurons, under the assumption that the neurons adopt the all-or-none law (McCulloch & Pitts, 1943).

The two-valued algebra was also a fundamental tool in the development of modern computers. A masters student at MIT named Claude Shannon realized that this algebra could be used to describe switching circuits (Shannon, 1938, 1949). In Shannon's use of the algebra, the symbol $\boldsymbol{X}$ represented a contact on a relay or a switch, and the symbol $\boldsymbol{\sim X}$ represented a break contact on a relay or switch. The series connection of two switches X and Y was represented by the expression $\boldsymbol{X} + \boldsymbol{Y}$, and the parallel connection of these two switches was represented by the expression $\boldsymbol{XY}$. A closed circuit was represented by the constant 0, and an open circuit was represented by the constant 1. Some of the modern variations of the two-valued algebra – such as Jevon's (1971) law of unity – meant that it could be used to simplify expressions far easier than could other algebras when they were applied to switching circuits. One could describe a circuit algebraically, and then manipulate this description to derive other equivalent circuits. One advantage of this is that the algebra could be used to derive simpler circuits.

A second advantage of Shannon's (1938) approach was that one could describe simple switching circuits – in particular, circuits in which there were two input lines and one output line – as a logic gate. Each input line represented a variable, which could be either true (i.e., 3 volts) or false (i.e., 0.5 volts). The logic gate combined these two inputs in a fashion that could be described in terms of one operator in the two-valued logic, and output a signal that could be interpreted as either being true or false. The table below illustrates four such logic gates for the two

inputs ***A*** and ***B***. One is an AND gate, one is an OR gate, one inverts the signal from ***A***, and one inverts the signal from ***B***.

Inputs		Logic Gates And Their Outputs			
A	B	AND	OR	INVERT A	INVERT B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False

A functionally complete set of gates is a set of logic gates that can be used to construct other circuits for computing any function in the two-valued algebra. The three types of logic gates represented in the table above (AND, OR, INVERT) represent a functionally complete set of gates.

11.3 PERCEPTRONS AND TWO-VALUED ALGEBRA

The purpose of the exercises in this chapter is to provide an introduction to training perceptrons using the Rosenblatt program. This will be accomplished by training a perceptron to compute the truth table for the four logic gates that was given above. The perceptron that will be trained will have two input units, one for the variable ***A***, the other for ***B***. If a variable is true, then its input unit will be activated with a value of 1. If a variable is false, then its input unit will be activated with a value of 0. There will be four different training patterns that will be presented to the perceptron, one for each possible combination of the values of ***A*** and ***B*** (i.e., one for each row in the table above).

The perceptron that will be trained will have four different output units. The output unit will be trained to generate an activation of 1 when its logical combination of ***A*** and ***B*** is true, and to generate an activation of 0 when its logical combination of ***A*** and ***B*** is false, as indicated in the table above. The first output unit will be trained to compute AND, the second to compute OR, the third to compute ***~A*** (i.e., to invert ***A***), and the fourth to compute ***~B***.

In our previous exercises that used the James program, when network error was used as a criterion to stop training, we did this by having training stop when the total sum of squared error for the network dropped below some minimum value. In the Rosenblatt program, a different approach is used. In the spirit of two-valued logic, when one pattern is presented to the network, each output unit is either going to be right or it is going to be wrong. When an output unit is right, we will count it as a “hit”. When an output unit is wrong, we will count it as a “miss”. The maximum number of hits that are possible for a training set is equal to the total number of training patterns times the total number of output units. For example, the network that we will be training below has four different training patterns, and four different output units. So, we will want training to stop when there are 16 hits (and 0 misses).

How is a hit defined? For each pattern presented during a training epoch, we will compute the squared difference between the desired activity of an output unit and its actual activity. When this squared difference is smaller than a minimum value that will be set before training, we will count a hit. Otherwise, we will count a miss. The default setting for this minimum value is 0.01. What this means is that if the desired output activity for a unit is 1, if it generates a value higher than 0.9 then it will count as a hit. This is because $(1 - 0.9)^2 = 0.1^2 = 0.01$, which is our minimum value. Using similar logic, if the desired output activity for a unit is 0, if it generates a value smaller than 0.1 then it will count as a hit. Otherwise, it will count as a miss. If for some reason we want to increase the accuracy of what a network learns, we can do this by decreasing this hit criterion. For instance, by setting it to 0.0025, the network will have to generate activity higher than 0.95 to turn “on”, and lower than 0.05 to turn “off”. If for some reason we want a more liberal definition of a hit, then we can do this by increasing this criterion value.

11.3.1 Procedure For The Delta Rule

Install and run the Rosenblatt program, referring back to Chapter 10 if necessary. Load the file "Boole4.net". On the setup page, choose the Delta rule, and keep the remaining settings at their default values:

- End after a maximum number of training epochs
- End when there are all "hits" and no "misses"
- Randomize patterns each epoch
- Train thresholds during learning
- Default starts for weights
- Default starts for thresholds
- Maximum number of epochs = 1000
- Number of epochs between printouts = 100
- Learning rate = 0.5
- Minimum level of squared error to define a "hit" = 0.01

Press the "Start Training" button to begin training. When the network converges to a solution to the problem, press the "Test Recall" button. Then, have the program build an Excel spreadsheet to summarize the results. This spreadsheet will contain all of the information required to answer the following questions. If you are using a version of the Rosenblatt program that does not use Excel, then save the results of training to a file that you can examine later to answer the questions.

11.3.2 Exercise 11.1

1. **What is the total SSE for the network after training has finished?**
2. **How many epochs of training occurred before the program stopped training the network?**
3. **When the Delta rule is used in the Rosenblatt program, the step activation function is being used in the output units. The unit will only turn on when its net input exceeds the unit's threshold or bias. Armed with this knowledge, look at the two connection weights that feed into the first output unit, and look at the bias of this output unit. Explain how this output unit computes the AND of *A* and *B*.**
4. **Look at the two connection weights that feed into the second output unit, and look at the bias of this output unit. Explain how this output unit computes the OR of *A* and *B*.**
5. **Look at the two connection weights that feed into the third output unit, and look at the bias of this output unit. Explain how this output unit INVERTS the signal from *A*.**
6. **Look at the two connection weights that feed into the fourth output unit, and look at the bias of this output unit. Explain how this output unit INVERTS the signal from *B*.**

11.3.3 Procedure For The Gradient Descent Rule

Run the Rosenblatt program once again on the file "Boole4.net". On the setup page, choose the gradient descent rule, and keep the remaining settings at their default values, which should be the same as in the previous exercise:

- End after a maximum number of training epochs
- End when there are all "hits" and no "misses"
- Randomize patterns each epoch
- Train thresholds during learning
- Default starts for weights

- Default starts for thresholds
- Maximum number of epochs = 1000
- Number of epochs between printouts = 100
- Learning rate = 0.5
- Minimum level of squared error to define a "hit" = 0.01

Press the "Start Training" button to begin training. If the network has not converged to a solution after 1000 epochs, press the "Continue Training" button. When the network converges to a solution to the problem, press the "Test Recall" button. Then, have the program build an Excel spreadsheet to summarize the results. This spreadsheet will contain all of the information required to answer the following questions. If you are using a version of the Rosenblatt program that does not use Excel, then save the results of training to a file that you can examine later to answer the questions.

11.3.4 EXERCISE 11.2

1. **What is the total SSE for the network after training has finished?**
2. **How many epochs of training occurred before the program stopped training the network?**
3. **How do your answers to questions 1 and 2 above compare to your answers to questions 1 and 2 in Exercise 11.1? If the answers are different, provide a brief explanation of why this is to be expected.**
4. **When the gradient descent rule is used in the Rosenblatt program, the logistic activation function is being used in the output units. Armed with this knowledge, look at the two connection weights that feed into the first output unit, and look at the bias of this output unit. Explain how this output unit computes the AND of *A* and *B*. How does this explanation compare to the explanation of AND that you provided for the perceptron that was trained with the Delta rule?**
5. **Look at the two connection weights that feed into the second output unit, and look at the bias of this output unit. Explain how this output unit computes the OR of *A* and *B*. How does this explanation compare to the explanation of OR that you provided for the perceptron that was trained with the Delta rule?**
6. **Look at the two connection weights that feed into the third output unit, and look at the bias of this output unit. Explain how this output unit INVERTS the signal from *A*. How does this explanation compare to the explanation of INVERT that you provided for the perceptron that was trained with the Delta rule?**

11.3.5 PROCEDURE FOR EXPLORING BIAS

Run the Rosenblatt program once again on the file "Boole4.net". On the setup page, choose the delta rule, and *set the option to hold thresholds constant during training*. Keep the remaining settings at their default values, which should be the same as in the previous exercise, with the exception of holding thresholds constant:

- End after a maximum number of training epochs
- End when there are all "hits" and no "misses"
- Randomize patterns each epoch
- *Hold thresholds constant during learning*
- Default starts for weights
- Default starts for thresholds
- Maximum number of epochs = 1000
- Number of epochs between printouts = 100
- Learning rate = 0.5
- Minimum level of squared error to define a "hit" = 0.01

Press the “Start Training” button to begin training. If the network has not converged to a solution after 1000 epochs, press the “Continue Training” button. If the network has not solved the problem after 3000 sweeps, then do not train any further. You should observe that the network generates only 13 hits, and 3 misses, after this amount of training, and that network performance will not improve. Have the program build an Excel spreadsheet to summarize the results. This spreadsheet will contain all of the information required to answer the following questions. If you are using a version of the Rosenblatt program that does not use Excel, then save the results of training to a file that you can examine later to answer the questions.

11.3.5 EXERCISE 11.3

1. **What is the total SSE for the network after training has finished?**
2. **How many epochs of training occurred before the program stopped training the network?**
3. **Examine the responses of the network to each pattern, and the errors computed for each output unit for each pattern. In what way is the network behaving correctly? In what way is the network making mistakes?**
4. **With the default settings, and with thresholds held constant during training, every output unit always has a threshold of 0. Armed with this knowledge, examine the connection weights that feed into any output unit that is generating errors. Explain why any errors are being made.**
5. **Given your answer to question 4, speculate on the role of the threshold in the perceptron, and speculate on why it might be important to let the learning rule train thresholds in addition to training the connection weights.**