CNS 188a Overview

• Boolean algebra as an axiomatic system

• Boolean functions and their representations using Boolean formulas and spectral methods

• Implementing Boolean functions with relay circuits, circuits of AON (AND, OR, NOT) gates and LT (Linear Threshold) gates

• Analyzing the complexity (size and depth) of circuits

• Relations (as opposed to functions) and their implementation in circuits

• Feedback and convergence in LT circuits
Basic Axioms and Properties

- **A1. Identities:**
  \[ a + 0 = a \quad \text{and} \quad a \cdot 1 = a \]

- **A2. Complements:**
  \[ a + \bar{a} = 1 \quad \text{and} \quad a \cdot \bar{a} = 0 \]

- **A3. Commutativity:**
  \[ a + b = b + a \quad \text{and} \quad a \cdot b = b \cdot a \]

- **A4. Distributivity:**
  \[ a + b \cdot c = (a + b) \cdot (a + c) \quad \text{and} \quad a \cdot (b + c) = a \cdot b + a \cdot c \]

- **T3. Associativity:**
  \[ (a + b) + c = a + (b + c) \]
  \[ (a \cdot b) \cdot c = a \cdot (b \cdot c) \]

- **T4. DeMorgan Laws:**
  \[ \overline{a + b} = \bar{a} \cdot \bar{b} \]
  \[ \overline{a \cdot b} = \bar{a} + \bar{b} \]

- **L1. Self Absorption:**
  \[ a + a = a \quad \text{and} \quad a \cdot a = a \]

- **L2: Simple Absorption:**
  \[ a + 1 = 1 \quad \text{and} \quad a \cdot 0 = 0 \]

- **T0. Duality:**
  Correctness is maintained when interchange + and \( \cdot \), as well as 0 and 1.

- **T1. Distinct Complement:**
  Every element has another element that is its unique complement.

- **T2. Absorption:**
  \[ a + ab = a \quad \text{and} \quad a \cdot (a + b) = a \]
Current State

**Representation Theorem (Stone 1936):**
Every *finite* Boolean algebra is *isomorphic* to a Boolean algebra of subsets of some *finite* set $S$.

**0-1 Theorem:**
An identity is true for any finite Boolean algebra if and only if it is true for a *two-valued* (0-1) Boolean algebra.

**DNF Representation Theorem:**
DNF is a representation: two Boolean functions are equal if and only if their DNFs is identical.
DNF Theorem

DNF Theorem:
Every Boolean function can be expressed in DNF.

\[(a \cdot \overline{b} + a \cdot c) + \overline{a} = (a \cdot \overline{b}) \cdot (a \cdot c) + \overline{a}\]
\[
= (\overline{a} + b)(\overline{a} + \overline{c}) + \overline{a}
\]
\[
= \overline{a} + b \cdot \overline{c} + \overline{a}
\]
\[
= \overline{a} + b \cdot \overline{c}
\]

Absorption:

• A2. Complements:
  \[a + \overline{a} = 1\text{ and } a \cdot \overline{a} = 0\]

• A4. Distributivity:
  \[a + b \cdot c = (a + b) \cdot (a + c)\text{ and } a \cdot (b + c) = a \cdot b + a \cdot c\]

\[
= \overline{a} \cdot (b + \overline{b}) \cdot (c + \overline{c}) + b \cdot \overline{c} \cdot (a + \overline{a})
\]
\[
= \overline{a} \cdot b \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot c + a \cdot \overline{b} \cdot \overline{c} + a \cdot b \cdot \overline{c} + \overline{a} \cdot b \cdot c
\]

• T4. DeMorgan Laws:
  \[
  \frac{(a + b)}{a \cdot b} = \overline{a} \cdot \overline{b}
\]
  \[
  \frac{(a \cdot b)}{a + b} = \overline{a} + \overline{b}
\]
DNF Theorem

**DNF Theorem:**
Every Boolean function can be expressed in DNF.

**Proof:** By the algorithm

1. Apply DeMorgan Theorem (**T4**) until each negation is applied to a single variable.
2. Apply **distributive axiom (A4)** to get a sum of terms.
3. Augment a missing variable \(a\) to a term using (**A2**) multiplying by \(a + \overline{a}\).
4. Use **self absorption (L1)** to eliminate duplicate terms.
Idea: construct a DNF by adding the normal terms that correspond to \( f=1 \).

\[
a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b
\]
**DNF from Look Up Tables**

**Use only the blue entries**

\[
a \oplus b = a \cdot \bar{b} + \bar{a} \cdot b
\]

\[
a_{x_i}^i = \begin{cases} 
a_i & \text{if } x_i = 1 \\
\bar{a}_i & \text{if } x_i = 0
\end{cases}
\]

\[
f(a_1, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(x_1, x_2, \ldots, x_n) a_1^{x_1} a_2^{x_2} \cdots a_n^{x_n}
\]
Boole (Shannon) Decomposition

\[ f(a_1, a_2, \ldots, a_n) = \bar{a}_1 \cdot f(0, a_2, \ldots, a_n) + a_1 \cdot f(1, a_2, \ldots, a_n) \]

**Proof:**

**DNF representation**

\[ f(a_1, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(x_1, x_2, \ldots, x_n) a_1^{x_1} a_2^{x_2} \cdots a_n^{x_n} \]

\[ 0^{x_1} = 1 \rightarrow x_1 = 0 \]

\[ f(0, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(x_1, x_2, \ldots, x_n) 0^{x_1} a_2^{x_2} \cdots a_n^{x_n} \]

\[ f(0, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(0, x_2, \ldots, x_n) a_2^{x_2} \cdots a_n^{x_n} \]
Boole (Shannon) Decomposition

\[ f(a_1, a_2, \ldots, a_n) = \bar{a}_1 \cdot f(0, a_2, \ldots, a_n) + a_1 \cdot f(1, a_2, \ldots, a_n) \]

\textbf{Proof:}

**DNF representation**

\[ f(a_1, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(x_1, x_2, \ldots, x_n) a_1^{x_1} a_2^{x_2} \cdots a_n^{x_n} \]

\[ f(0, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(0, x_2, \ldots, x_n) a_2^{x_2} \cdots a_n^{x_n} \]

\[ \bar{a}_1 \cdot f(0, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(0, x_2, \ldots, x_n) \bar{a}_1 \cdot a_2^{x_2} \cdots a_n^{x_n} \]

\[ \text{DNF part that corresponds to } x_1 = 0 \]
Boole (Shannon) Decomposition

\[ f(a_1, a_2, \ldots, a_n) = \bar{a}_1 \cdot f(0, a_2, \ldots, a_n) + a_1 \cdot f(1, a_2, \ldots, a_n) \]

**Proof:**

**DNF representation**

\[ f(a_1, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(x_1, x_2, \ldots, x_n)a_1^{x_1}a_2^{x_2} \cdots a_n^{x_n} \]

\[ \bar{a}_1 \cdot f(0, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(0, x_2, \ldots, x_n)\bar{a}_1 \cdot a_2^{x_2} \cdots a_n^{x_n} \]

**DNF part**

\[ x_1 = 0 \]

\[ a_1 \cdot f(1, a_2, \ldots, a_n) = \sum_{X \in \{0,1\}^n} f(1, x_2, \ldots, x_n)a_1 \cdot a_2^{x_2} \cdots a_n^{x_n} \]

**DNF part**

\[ x_1 = 1 \]

Q
Why is the Decomposition Important?

Idea: can help in compressing representations of Boolean functions (Decision Trees, Binary Decision Diagrams (BDD)) Key in design, optimization and verification of circuits.

<table>
<thead>
<tr>
<th>ab</th>
<th>XOR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b \]

![Decision Tree Diagram]
Why is the Decomposition Important?

Decomposition results in a Binary Decision Tree

\[ f(a_1, a_2, \ldots, a_n) = \overline{a}_1 \cdot f(0, a_2, \ldots, a_n) + a_1 \cdot f(1, a_2, \ldots, a_n) \]

\[ \begin{align*}
0 &= \overline{f(a_1, a_2, \ldots, a_n)} \\
1 &= f(0, a_2, \ldots, a_n) \end{align*} \]
"The further back you look, the further forward you can see"

Winston Churchill

Gottfried Leibniz
1646-1716

- Logic and binary system
- Calculus

George Boole
1815-1864

Connected logic with algebra, 1847

Edward Huntington
1874-1952

Boolean axiomatic system, 1904

Shannon
1916-2001

Boolean algebra to electrical circuits, 1938

Marshal Stone
1903-1989

Boolean representation theorem, 1936
And accordingly, numbers are expressed as follows:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
</tr>
<tr>
<td>10000</td>
<td>16</td>
</tr>
<tr>
<td>10001</td>
<td>17</td>
</tr>
<tr>
<td>10010</td>
<td>18</td>
</tr>
<tr>
<td>10011</td>
<td>19</td>
</tr>
<tr>
<td>10100</td>
<td>20</td>
</tr>
<tr>
<td>10101</td>
<td>21</td>
</tr>
<tr>
<td>10110</td>
<td>22</td>
</tr>
<tr>
<td>10111</td>
<td>23</td>
</tr>
<tr>
<td>11000</td>
<td>24</td>
</tr>
<tr>
<td>11001</td>
<td>25</td>
</tr>
<tr>
<td>11010</td>
<td>26</td>
</tr>
<tr>
<td>11011</td>
<td>27</td>
</tr>
<tr>
<td>11100</td>
<td>28</td>
</tr>
<tr>
<td>11101</td>
<td>29</td>
</tr>
<tr>
<td>11110</td>
<td>30</td>
</tr>
<tr>
<td>11111</td>
<td>31</td>
</tr>
<tr>
<td>100000</td>
<td>32</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>
Leibniz – Binary System

Binary addition algorithm

§72  As for addition, it is simply done by counting and making periods when there are numbers to add together, adding up each column as usual, which will be done as follows: count the unities of the column, for example, for 29, look how this number is written in the table, to wit, by 11101; thus you write 1 under the column and put periods under the second, third and fourth column thereafter. These periods denote that it is necessary to count out one unity further in the column following.
Wen Wang (who flourished in about 1150 BC) is traditionally thought to have been author of the present hexagrams.
Connection Between
Boolean Calculus and Physical Circuits
Shannon 1938

The basic framework:
• A graph with edges and vertices
• Source and destination vertices (S and D)
• Boolean variables control the flow in the edges
• The function is 1 if S and D are connected
• The function is 0 if S and D are disconnected
Connection Between
Boolean Calculus and Physical Circuits
Shannon 1938

Q: How to design an AND gate?

\[ a \cdot b \]
Connection Between Boolean Calculus and Physical Circuits
Shannon 1938

Q: How to design an OR gate?

\[ a + b \]
Every 0-1 Boolean Function Can be Implemented Using A Depth Two Circuit

How?

Implement the DNF representation: $OR$ of many $ANDs$
Q: relay circuits to Boolean calculus?

connection between Boolean calculus and physical circuits
Shannon 1938

\[ f = ab + de + ace + dcb \]
Connection Between
Boolean Calculus and Physical Circuits
Shannon 1938

Boolean functions ↔ Relay circuits
Every 0-1 Boolean Function Can be Implemented Using A Depth Two Circuit

How?

Implement the DNF representation: \( OR \) of many \( ANDs \)

Depth: longest path from input to output - counting the number of gates

Size: total number of gates in the circuit
XOR of Two Variables

\[ a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b \]

Depth = 2

Size = 3
XOR of two Variables with Relays?

\[ a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b \]

We need to use 6 relays
Two relays per gate
XOR of two Variables with Relays?

\[ a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b \]

Q: Can we compute XOR with less than 6 relays?
XOR of More Variables

\[ \text{XOR}(x_1, x_2, \ldots, x_n) = \begin{cases} 
0 & \text{if } |X| \text{ (number of 1's in } X) \text{ is even} \\
1 & \text{if } |X| \text{ is odd} 
\end{cases} \]

\[ a \oplus b \oplus c = \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot \overline{c} + a \cdot b \cdot c \]

\[ \begin{array}{c|c}
\text{abc} & \text{XOR}(a,b,c) \\
\hline
000 & 0 \\
001 & 1 \\
010 & 1 \\
011 & 0 \\
100 & 1 \\
101 & 0 \\
110 & 0 \\
111 & 1 \\
\end{array} \]

How many gates for XOR with AON?

\[ \text{AON} = \text{AND} \text{ OR} \text{ NOT} \]
XOR of Three Variables with AON

Depth = 2
Size = 5

\[ a \oplus b \oplus c = \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot b \cdot c \]

• is the complement
**XOR of Four Variables with AON**

**Depth = 2**

**Size = 9**

4 ANDs for \(|X| = 1\)

4 ANDs for \(|X| = 3\)

Can we do better?

No

Yes

Size 9 optimal for depth 2

How?
Size 8 XOR AON Circuit of Four Variables

optimal size!
Neuron - Neural Gate

Linear Threshold (LT) gate

\( f(x) \in \{0, 1\} \)

\[ F(X) = w_0 + \sum_{i=1}^{n} w_i x_i \]

\[ f(X) = \text{sgn}(F(X)) = \begin{cases} 
0 & \text{if } F(X) < 0 \\
1 & \text{if } F(X) \geq 0
\end{cases} \]
Can We Compute an AND Function with an LT Gate?

\[ AND(x_1, x_2) \]

\[ F(x_1, x_2) = -2 + x_1 + x_2 \]

\[ F(X) = w_0 + \sum_{i=1}^{n} w_i x_i \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( F(X) )</th>
<th>( f(X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Can We Compute an OR Function with an LT Gate?

\[ \text{OR}(x_1, x_2) \]

\[ F(x_1, x_2) = -1 + x_1 + x_2 \]

\[ F(X) = w_0 + \sum_{i=1}^{n} w_i x_i \]

<table>
<thead>
<tr>
<th>( x_1 \times x_2 )</th>
<th>( F(X) )</th>
<th>( f(X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Can We Compute a NOT with an LT Gate?

Can we compute \( \text{NOT} \) without \( sgn \)?

\[
\bar{x}_1 = \text{NOT}(x_1) = 1 - x_1
\]
More Variables for AND?

\[ F(x_1, x_2) = -2 + x_1 + x_2 \]

\[ F(x_1, x_2, \ldots, x_n) = -n + x_1 + x_2 + \cdots + x_n \]

\[ |X| = \sum_{i=1}^{n} x_i \]

\[ -n + |X| \]

\[ F(X) = \begin{cases} 
0 & \text{if } |X| = n \\
< 0 & \text{if } |X| < n 
\end{cases} \]

Hence \( sgn(F(X)) \) is an AND
More Variables for OR?

\[ F(x_1, x_2) = -1 + x_1 + x_2 \]

\[ F(x_1, x_2, \ldots, x_n) = -1 + x_1 + x_2 + \cdots + x_n \]

\[ |X| = \sum_{i=1}^{n} x_i \quad -1 + |X| \]

\[ F(X) = \begin{cases} 
-1 & \text{if } |X| = 0 \\
\geq 0 & \text{if } |X| \geq 1
\end{cases} \]

Hence \( \text{sgn}(F(X)) \) is an OR
**AON vs LT Circuits**

**Theorem (LT/AON):**

LT circuits are at least as powerful as AON circuits:

(i) **Size** of an AON circuit for a Boolean function $f$ is greater/equal size of an optimal LT circuit for the function $f$

(ii) **Depth** of an AON circuit for a Boolean function $f$ is greater/equal depth of an optimal LT circuit for the function $f$
Theorem (LT/AON):

LT circuits are at least as powerful as AON circuits:

(i) Size of an AON circuit for a Boolean function $f$ is greater/equal size of an optimal LT circuit for the function $f$.

(ii) Depth of an AON circuit for a Boolean function $f$ is greater/equal depth of an optimal LT circuit for the function $f$.

Proof:

By the previous reductions: $AND$, $OR$ and $NOT$ can be implemented by a single LT gate.
LT gates are MORE Powerful

XOR of three variables with AON
- Depth = 2
- Size = 5

XOR of four variables with AON
- Depth = 2
- Size = 9

Size is optimal for depth 2

XOR of four variables with AON, size 8 is optimal for any depth
LT gates are MORE Powerful

Size 5 is optimal for AON depth 2

Size 4 LT depth 2

\[ XOR(x_1, x_2, x_3) \]
LT gates are MORE Powerful

LT-l = LT layered, inputs go to first layer only

<table>
<thead>
<tr>
<th>X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A+B+C</th>
<th>-2+A+B+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

LT gates are MORE Powerful

Can take the sgn or add 1
LT gates are EVEN MORE Powerful

LT-nl = LT non-layered, inputs go to any layer

\[ \text{LT-nl} = \text{LT non-layered, inputs go to any layer} \]

\[
\begin{array}{c|c|c|c}
|X| & A & -1-2A+|X| & \text{sgn}( ) \\
0 & 0 & -1 & 0 \\
1 & 0 & 0 & 1 \\
2 & 1 & -1 & 0 \\
3 & 1 & 0 & 1 \\
\end{array}
\]

Can take the sgn or add 1
### XOR Function: Size of LT vs AON in Depth 2

<table>
<thead>
<tr>
<th></th>
<th>$\text{XOR}(x_1, x_2, x_3)$</th>
<th>$\text{XOR}(x_1, x_2, \ldots, x_n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AON</td>
<td>5</td>
<td>$2^{n-1} + 1^*$</td>
</tr>
<tr>
<td>LT-I</td>
<td>4</td>
<td>$n + 1^*$</td>
</tr>
<tr>
<td>LT-nl</td>
<td>2</td>
<td>$\left\lfloor \frac{n}{2} \right\rfloor + 1^*$</td>
</tr>
</tbody>
</table>

* = it is optimal

Exponential gap in size
Is it possible to compute \( \text{XOR}(x_1, x_2) \) with a single LT element?

Is this result general? more variables?
XOR with a Single LT Gate

Is it possible to compute $XOR(x_1, x_2)$ with a single LT element?

Answer: NO

Proof: By contradiction assume it is possible and reach a contradiction

$XOR(x_1, x_2) = sgn(w_0 + w_1x_1 + w_2x_2)$

- $x_1 = 0$, $x_2 = 0$ \( \Rightarrow w_0 < 0 \)
- $x_1 = 1$, $x_2 = 0$ \( \Rightarrow w_0 + w_1 \geq 0 \)
- $x_1 = 0$, $x_2 = 1$ \( \Rightarrow w_0 + w_2 \geq 0 \)
- $x_1 = 1$, $x_2 = 1$ \( \Rightarrow w_0 + w_1 + w_2 < 0 \)

Contradiction

\[ 2w_0 + w_1 + w_2 < 0 \]

\[ 2w_0 + w_1 + w_2 \geq 0 \]
Symmetric Functions

AND, OR and XOR are symmetric Boolean functions

Permuting the inputs does not change the output

\[ XOR(x_1, x_2, x_3) = XOR(x_2, x_1, x_3) = XOR(x_2, x_3, x_1) \]

Definition: A Boolean function \( f \) is symmetric if

\[ f(X) = f(\pi(X)) \]

for an arbitrary permutation \( \pi \)
Questions on Symmetric Functions

AND, OR and XOR are symmetric Boolean functions

Q1: How many symmetric Boolean functions of n variables?

Q2: What are the symmetric functions that can be realized by a single LT gate?
History on LT

Boolean representation theorem, 1936

Warren McCulloch
1899 - 1969

Computing with LT (neural) gates: connection between circuits and neural networks, 1943

Walter Pitts
1923 - 1969

Boolean algebra to electrical circuits, 1938

Marshal Stone
1903-1989

Claude Shannon
1916-2001
History on LT

Warren McCulloch
1899 - 1969

Walter Pitts
1923 - 1969

Warren McCulloch arrived in early 1942 to the University of Chicago, invited Pitts, who was still homeless, to live with his family.

In the evenings McCulloch and Pitts collaborated. Pitts was familiar with the work of Gottfried Leibniz on computing and they considered the question of whether the nervous system could be considered a kind of universal computing device as described by Leibniz.

This led to their 1943 seminal neural networks paper: *A Logical Calculus of Ideas Immanent in Nervous Activity.*