Valerio Mante, Matthew Cook, Benjamin Grewe, Giacomo Indiveri, Daniel Kiper, Wolfger von der Behrens
Lecture 11
Asst: Ioana Calangiu, Raphaela Kreiser, Nicoletta Risi {calangiu@ini.uzh.ch, rkreiser@student.ethz.ch, risi@ini.uzh.ch}

Solution 11.1: The McCulloch-Pitts Neuron

1. This question is formulated very openly and it is possible to come up with very many answers. Here are a few relatively relevant ones. Similarities:

   - Biological and McCulloch-Pitts neurons can be active or inactive.
   - They have a directionality (input/output).
   - The activation of a neuron is dependent on a weighted function of other neurons.

   Differences:

   - Real neurons exist in continuous time, whereas McCulloch-Pitts neurons operate in discrete time.
   - Real neurons have degrees of activation, not just on/off states.
   - The activation as a function of the inputs of a real neuron (typically) is not linear (or thresholded linear).

2. Realizable are NAND and AND, not realizable are XOR and NOT(XOR). To see this, imagine the Perceptron as drawing a straight line in the 2D input space (in n dimensions a McCulloch-Pitts Neuron draws a $n-1$ dimensional hyperplane). A successful 'classification' of the inputs means that all inputs of any given value lie on the same side of this line. Arrangements of points that cannot be separated by a line correspond to mappings that cannot be realized by the Perceptron; the XOR and NOT(XOR) are examples of such mappings.

3. (a) Does not change realizability.
   (b) The parameters of a McCulloch-Pitts neuron can be rescaled by a common factor; the relevant numbers are their ratios to the threshold. Using integers these ratios are always rational numbers, which can approximate any real number with arbitrary precision. The same functions can thus be implemented.
   (c) In this case the separation line will always intersect the origin. This does affect the possible classifications. If it is allowed to add an input that is always positive, one can get back to the case with threshold.

Solution 11.2: Computing Parity

A. Linear Threshold Gates

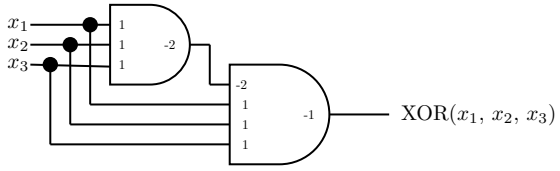

Figure 1: A circuit for XOR of three inputs, as shown in class.

$$y = \begin{cases} 1 & \text{if} \quad w_0 + \sum_{i=1}^{n} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Applying Equation 1 to the circuit in Figure 1 gives the following output (y_2 is the output of Gate 2).

$\sum_i x$	y_2	XOR(x_1, x_2, x_3)
0	-1	0
1	0	1
2	-1	0
3	0	1

Table 1: Output of circuit in Figure

Why do we need a bias of -1? If we look at the output of the circuit before subtracting the bias -1, then y_2 would be either 0 or 1 or 0 or 1 when the sum of the inputs is either 0 or 1 or 2 or 3. According to Equation 1, 0 and 1 both correspond to ON states. Thus, subtracting -1 with the second gate, the circuit is able to generate both ON and OFF states.

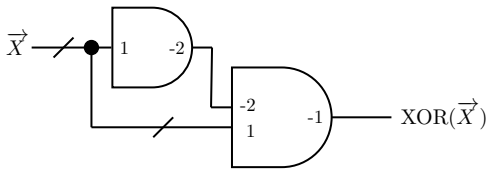

Figure 2: The same circuit as in figure 1, but here the three inputs are shown as the vector $\overrightarrow{X}$, and the three wires are shown as a single line with a diagonal mark on it. Where this "wire bundle" goes into a gate, the weight shown is used for every wire in the bundle.

1. The circuit in figure 2 is a valid circuit for XOR if $\overrightarrow{X}$ has dimensionality 2.
2. The circuit in figure 2 is *not* a valid circuit for XOR if $\overrightarrow{X}$ has dimensionality 4.

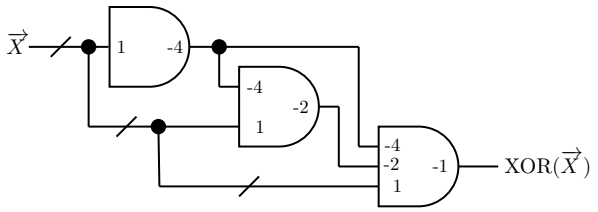

Figure 3: Here is a bigger XOR circuit.

3. The circuit in figure 3 can process $2^3 - 1$ inputs reliably.

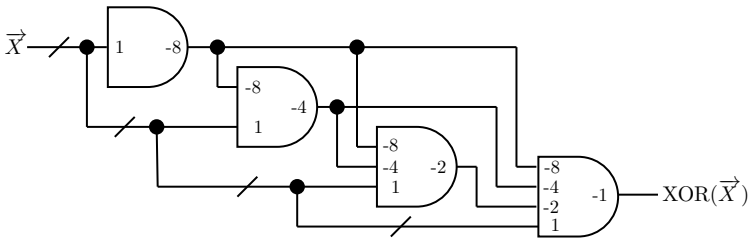

Figure 4: Here is an even bigger XOR circuit.

4. The circuit in figure 4 can process $2^4 - 1$ inputs reliably.
5. Figure 5 is the next circuit in this pattern.
   The circuit in figure 5 can process $2^5 - 1$ inputs reliably.

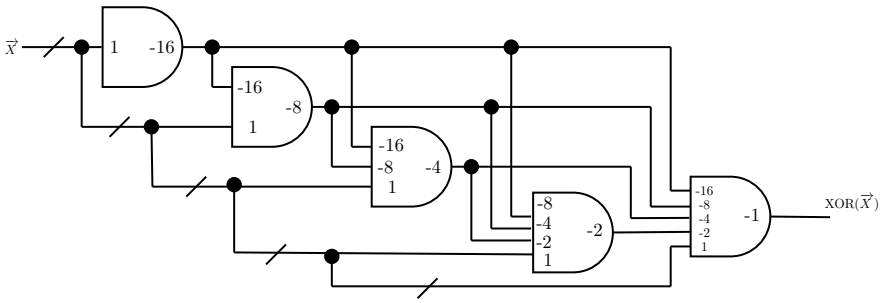

Figure 5: Here is the next circuit, a XOR with 5 gates.

6. So far we saw that n inputs need $2^x - 1$, where x is the number of gates. Thus, we need $floor[log_2 n] + 1$ gates. (Note: For people who were at the tutorial, I said you would round $log_2 n$, but it's wrong, you floor the logarithm.)

7. XOR checks the parity of the sum of the inputs. We therefore have two possible scenarios.

   Case 1. $\sum_i x_i$ is even. Adding a NOT to one of the inputs will change the parity of the sum (to odd) and the circuit $\text{XOR}(x_1, x_2, ..., x_{n-1}, x_n)$ will output 1. Adding a NOT to the output, will result in an output 0. This is the correct answer for a set of inputs with even sum.

   Case 2. Follows the same argument, but you start with $\sum_i x_i$ odd.

$$\text{XOR}(x_1, x_2, ..., x_{n-1}, x_n) = \overline{\text{XOR}(x_1, x_2, ..., x_{n-1}, \overline{x_n})} \tag{2}$$

   where the vertical line indicates the presence of a NOT gate.

8.
$$\text{XOR}(x_1, x_2, ..., x_{n-1}, x_n = 0) = \text{modulo}_2(\sum_i^{n-1} x_i + 0) = \text{XOR}(x_1, x_2, ..., x_{n-1}) \tag{3}$$

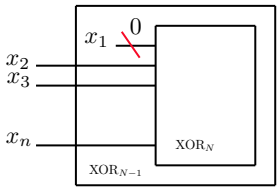

Figure 6: If you change one of the inputs to always be 0 instead of being an input, then the resulting circuit computes XOR of one fewer inputs.

B. AND/OR/NOT (AON) Gates

9. Consider two cases. **Case 1** - The 0 input goes directly into an AND gate. This will set the output of the AND gate to 0, no matter how many other x_i inputs are connected to this AND gate. Thus, we can bypass this AND gate, and just propagate the 0 input further into the circuit. **Case 2** - The 0 input goes directly into an OR gate. This doesn't affect the OR gate in any way, since we still have to look at all

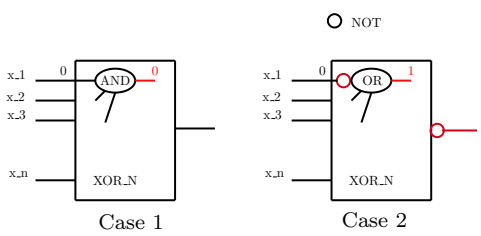

Figure 7: If you change an input to always be 0, then you can simplify the circuit (without affecting its output) so that it has at least one fewer AND or OR gates

the other x_i inputs and find one that is 1. However, if we introduce a NOT gate after the input 0, we will make sure the output of the OR gate is always 1, so we can bypass the OR gate and send 1 into the circuit. According to the result of exercise 7, we need to add another NOT at the output of the XOR circuit, to not alter its function.

If the first gate connected to the 0 input would be a NOT (or a sequence of NOTs), we would look at the first non-NOT gate.

10. Given a large circuit of M gates and N inputs, we can test its ability to compute XOR, by iteratively applying 8 and 9. We set an input to be 0, and then remove on logic gate. After the first step, we are left with a circuit of M-1 gates and N-1 inputs. If M<N, then, at some point, we would be left with a circuit with several inputs and no gates. Thus, one needs M≥N. (Note: In fact, it has been shown that the number of gates M needs to be much higher than the number of inputs N).

11. To compute XOR on N inputs one needs $log_2(N)$ LT gates (6), or N AON gates (10).

$$\begin{aligned} \text{Complexity}_{\text{AON}}(N) &= N \\ \text{Complexity}_{\text{LT}}(N) &= log_2(N) \\ \text{Complexity}_{\text{AON}}(N) &= 2^{\text{Complexity}_{\text{LT}}(N)} \end{aligned} \tag{4}$$