Lecture 3: Tight Bounds for Circuits and Counting Arguments

Anup Rao

October 4, 2018

In today's lecture we discuss a matching upper bound and lower bound for boolean circuits. On the one hand, we shall prove that every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n/n)$, and on the other hand we show that for n large enough there is a function that cannot be computed by a circuit of size less than $2^n/(3n)$.

The lower bound—Counting arguments

The lower bound we prove here was first shown by Shanon. He introduced a really simple but powerful technique to prove it, called a *counting argument*.

Theorem 1. *For every large enough n, there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/3n$.*

Proof We shall count the total number of circuits of size s, where $s > n$. To define a circuit of size s, we need to pick the logical operator for each (non-input) gate, and specify where each of its two inputs come from. So the number of choices for each non-input gate is at most $3s^2$. The number of choices for an input gate is at most $n < 3s^2$. So, the total number of choices for each gate is at most $3s^2 + n$, and the number of possible circuits of size s is at most

$$(3s^2 + n)^s \leq (4s^2)^s = 2^{s \log(4s^2)} < 2^{3s \log s},$$

when $n > 4$.

This means that the total number of circuits of size $2^n/3n$ is less than $2^{3 \cdot \frac{2^n}{3n} \cdot n} = 2^{2^n}$. On the other hand, the number of functions $f : \{0,1\}^n \to \{0,1\}$ is exactly 2^{2^n}. Thus, not all these functions can be computed by a circuit of size $2^n/(3n)$. ■

Indeed, the above argument shows that the fraction of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by a circuit of size $2^n/4n$ is at most $\frac{2^{\frac{3}{4} \cdot 2^n}}{2^{2^n}} = \frac{1}{2^{2^{n-2}}}$, which is extremely small.

Similar arguments can be used to show that not every function has an efficient branching program (as you will do on your homework).

Not every function has an efficient communication protocol

LET US SEE another example of how counting arguments can be used to prove lower bounds. We didn't discuss this example in class, but I add it here to illustrate how flexible counting is.

Recall that a communication protocol for computing a function $f(x, y)$ specifies a way for Alice and Bob to communicate with each other in order to compute f. If x, y are n-bit strings, the number of such functions f is $2^{2^{2n}}$: indeed there are 2^{2n} inputs x, y, and for each choice of input, there are 2 choices for the output of the function.

The trivial protocol for computing an arbitrary function is to have Alice send her entire input to Bob. This takes n bits of communication. Here we show:

Theorem 2. *There is a function f that requires at least $n-2$ bits of communication.*

Proof As with circuits, we do the proof by counting. To count the number of communication protocols, observe that for every prefix of messages communicated so far, there are 2 choices for who should send the next bit, there are 2^{2^n} choices for the function to use to send that next bit.

If the communication complexity is at most t bits, the number of strings of length at most t is at most 2^{t+1}, so the number of protocols is thus $\left(2 \cdot 2^{2^n}\right)^{2^{t+1}} = 2^{(2^n+1)2^{t+1}} \leq 2^{2^{n+t+2}}$. So if $t < n-2$, the number of protocols is strictly less than the number of functions f. ■

The upper bounds on Circuit Size

We shall prove that *every* function can be computed in size $O(2^n/n)$. To work up to the proof, we start by giving some weaker bounds.

A formula in disjunctive normal form (DNF) is a formula that can be written as an OR of AND's. Suppose we have a function $f : \{0,1\}^3 \to \{0,1\}$ that takes the value 1 on 4 inputs: 000, 111, 010 and 101. Then f can be computed using this formula:

$$(\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3)$$

Each term in the formula outputs 1 on exactly one input. The same idea can be used to prove:

Theorem 3. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a DNF of size at most $O(n \cdot 2^n)$.*

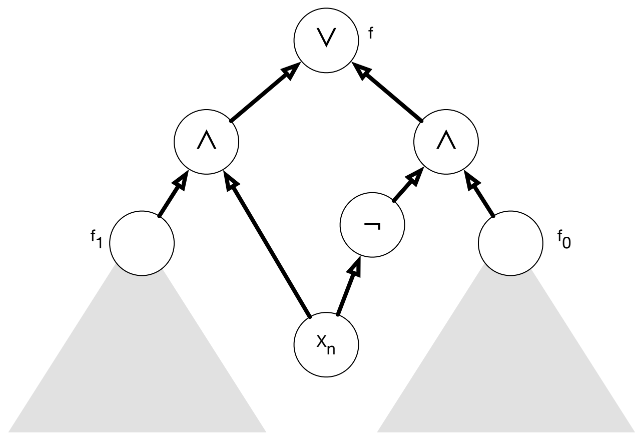

Figure 1: Recursive construction of a circuit for f.

Proof For every input $y \in \{0,1\}^n$, we can define a term F_y of size $O(n)$ that outputs 1 only when $x = y$. The term is the AND of all variables, where the i'th variable is negated if and only if $y_i = 0$. f can then be computed as the OR of all F_y for which $f(y) = 1$. ■

To get a better bound, we need to use a lot more alternation between AND and OR gates. We can use a recursive construction to prove:

Theorem 4. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n)$.*

Proof We construct the circuit recursively. When $n = 1$, there is clearly a constant sized circuit that computes f, since f must be either a constant, x_1 or $\neg x_1$.

For $n > 1$, let f_0 denote the function on $n-1$ bits given by $f_0(x) = f(x,0)$, and $f_1(x) = f(x,1)$. Then by induction we can compute f_0, f_1 recursively, and combine them using the value of the last bit to obtain f, as in Figure 1. When $x_n = 1$, the circuit outputs $f_1(x_1, \ldots, x_{n-1})$, and when $x_n = 0$, the circuit outputs $f_0(x_1, \ldots, x_{n-1})$.

If S_n is the size of the resulting circuit when the underlying function takes an n bit input, we have proved that

$$S_n \leq 2S_{n-1} + 5.$$

Expanding this recurrence, and using the fact that $S_1 \leq 5$, we get that

$$S_n \leq \sum_{i=1}^{n} 2^i 5 = 5 \cdot (2^{n+1} - 1) < 10 \cdot 2^n,$$

where here we used the formula for computing the sum of a geometric series. ■

Finally, we add one more idea to shave off another factor of n:

Lupanov proved this.

Theorem 5. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed in size $O(2^n/n)$ and depth $O(n)$.*

Proof We will use a recursive construction, but stop the recursion at a certain point. For a parameter t, we start by computing *every* function of the first t bits of the input. There are 2^{2^t} such functions, and each one can be computed by a circuit of size $O(2^t)$ using Theorem 4, so we can compute every function using at most $O(2^t \cdot 2^{2^t}) \leq O(2^{t+2^t})$ gates.

To compute the function f, we use the recursive construction defined in the proof of Theorem 4 for $n-t$ steps. After $n-t$ steps, we have put down $O(2^{n-t})$ gates, and need to compute functions on the first t bits, but since we have already computed every such function, we are done. The size of the final circuit is thus $O\left(2^{t+2^t} + 2^{n-t}\right)$. Setting $t = \log n - 1$, we get a circuit of size

$$O(2^{\log n - 1 + n/2} + 2^{n - \log n + 1}) \leq O(2^n/n).$$

■