CSE 513T Theory of Artificial Intelligence & Machine Learning January 25, 2018

Lecture 4

Lecturer: Brendan Juba *Scribe: Ray Lo*

Contents

1 Circuits and NP-Completeness **1**
1.1 Definition "Boolean Circuit" 1
1.2 Cook-Levin Theorem 2
1.3 3-SAT is NP-Complete 3

2 Proper learning versus improper learning **4**

3 Proper learning solves Consis **4**

1 Circuits and NP-Completeness

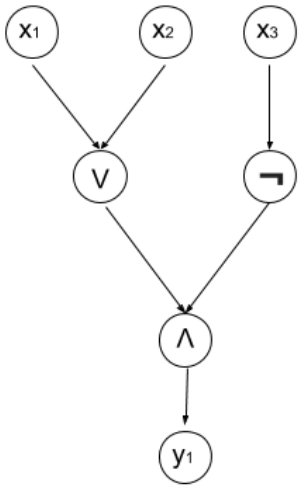
1.1 Definition "Boolean Circuit"

A Boolean Circuit can be thought of as function f:

$$f : \{0,1\}^n \to \{0,1\}^k$$

The representation of a Boolean Circuit is an acyclic directed graph.

- The "source" nodes, nodes with only outgoing edges and no incoming edges, are labeled x_1 through x_n or constants 0 or 1.
- The k "sink" nodes, nodes with only incoming edges and no outgoing edges, labeled y_1 through y_k.
- The "internal" nodes, nodes that have both edges arriving in and out, represent logic gates. Specifically, there are three types of inner nodes:
  - "OR"- has two inputs and returns the inclusive "OR" of the two inputs as its output(s).
  - "AND" - has two inputs and returns the "AND" of the two inputs as its output(s).
  - "NOT" - has one input and returns the negation of the input as its output(s).



An example of a Boolean Circuit

1.2 Cook-Levin Theorem

Given the following definition of CIRCUIT-SAT:

$$\text{CIRCUIT-SAT} = \{(c, \overline{x}) : c \in \text{BOOL-CIRCUIT}, c(\overline{x}) = 1\}$$

In other words, CIRCUIT-SAT asks us to find a set of "settings" for the input binary vector $\overline{x}$ such that the Boolean Circuit c outputs 1.

The Cook-Levin Theorem states that: ***CIRCUIT-SAT is NP-Complete***

Proposition Before we prove the Cook-Levin, we consider the following proposition.

Suppose there exists a polynomial-time algorithm computing some function:

$$f = \{f_n : \{0,1\}^n \to \{0,1\}^{k(n)}\}_{n=1}^{\infty}$$

The proposition states that then there exists an algorithm, that on input integer n, runs in polynomial time in n, that outputs a boolean circuit c_n, s.t.

$$\forall \mathbf{x} \in \{0,1\}^n, c_n(\mathbf{x}) = f_n(\mathbf{x})$$

In other (looser) words, the proposition states that it's possible to efficiently turn an efficient algorithm, which takes in some boolean vector and outputs some other boolean vector, into a boolean circuit.

We did not prove this proposition in class. The big idea for such proof would be to "unroll" the main loop of our given algorithm some poly(n) times.

Observation the proposition allows us to connect circuits to algorithms. The theorem indicates that polynomial-sized boolean circuits are very expressive.

More on the proof of Cook-Levin Let S be any NP-Search problem.

Since S is efficiently verifiable by definition, the proposition suggests that in polynomial time, we can turn this polynomial-time verification algorithm into a boolean circuit C_s, such that this circuit computes:

$$S(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & (\mathbf{x}, \mathbf{y}) \in S \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Now, what we would like to do is to find the appropriate $\mathbf{y}$ that would solve our search problem given $\mathbf{x}^*$.

To do so, we can hardwire the $\mathbf{x}$ inputs of this boolean circuit c_s to be $\mathbf{x}^*$.

We then only need to use the polynomial balance condition of S (the second part in the definition of an NP-Search problem) to generate the circuit for S on inputs of total length:

$$n' = |\mathbf{x}^*| + |poly(|\mathbf{x}^*|)|$$

where the second input y may be padded to length exactly $poly(|\mathbf{x}^*|)$

We have turned c_s into $c_{\mathbf{x}^*}(\mathbf{y})) = S(\mathbf{x}^*, \mathbf{y})$

We can now pass $c_{\mathbf{x}^*}(\mathbf{y}))$ into our CIRCUIT-SAT subroutine and return the $\mathbf{y}$ it finds as the output. Since we made the second input ($\mathbf{y}$) large enough, if there exists a $\mathbf{y}$ such that $(\mathbf{x}^*, \mathbf{y}^*) \in S$, then the CIRCUIT-SAT subroutine will return $(\mathbf{x}, \mathbf{y}^{**}) \in S$, so $\mathbf{y}^{**}$ is a valid input for $\mathbf{x}^*$

Of course, we can efficiently verify that $\mathbf{y}^{**}$ is indeed a correct answer for $\mathbf{x}^*$ before returning it, as S is verifiable in polynomial time.

We have now shown that for any arbitrary NP-Search problem S, it's possible to efficiently use CIRCUIT-SAT as a subroutine to solve S. By definition, this make CIRCUIT-SAT an NP-Complete problem. ■

1.3 3-SAT is NP-Complete

Definition of 3-SAT

$$\text{3-SAT} = \{(\tau, \overline{x})\tau \text{ is 3CNF}, \tau(x) = 1\}$$

A 3CNF is an AND of ORs of at most 3 literals, such as:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_4)...$$

NP-Completeness Proof We will show that 3-SAT is NP-Complete by solving CIRC-SAT using 3-SAT as a subroutine.

Reduction:

- We create a new variable for each wire of the boolean circuit, in addition to the input and output variables.
- For each internal "AND gate" nodes, we write down a list of clauses that encodes the truth table. For instance, for an AND gate with input wires w_i and w_j, and output wire w_k:

  1. $(w_i \wedge w_j) \implies w_k$
  2. $(w_i \wedge \neg w_j) \implies \neg w_k$
  3. $(\neg w_i \wedge w_j) \implies \neg w_k$
  4. $(\neg w_i \wedge \neg w_j) \implies \neg w_k$

  Having written out the truth table, we can turn the four clauses into four 3CNF clauses, using the identity that $A \implies B \equiv \neg A \vee B$

  1. $\neg w_i \vee \neg w_j \vee w_k$
  2. $w_i \vee \neg w_j \vee \neg w_k$
  3. $\neg w_i \vee w_j \vee \neg w_k$
  4. $w_i \vee w_j \vee w_k$

- We may use a similar procedure for OR gates.
- For NOT gates, we write down:

  1. $w_i \implies \neg w_j$
  2. $\neg w_i \implies w_j$

  Again, using the same identity, we turn the two "implies" clauses into two 3CNF clauses:

  1. $\neg w_i \vee \neg w_j$
  2. $w_i \vee w_j$

- Finally, we add y itself as a clause, this forces the output of the circuit to be 1.

Correctness: Given any setting of the input wires x_1 through x_n, by induction on the depth of the gates in the circuit, all but one of the clauses for that gate will be trivially satisfied, since the inputs will be uniquely determined to be values that don't match the line in the truth table and the one remaining line uniquely determines the output wire's value. Therefore, since any satisfying assignment is forced to satisfy $y = 1$, the $x_1...x_n$ variables must be an input causing the circuit to output 1. We return this. ■

2 Proper learning versus improper learning

Recall that we defined PAC-Learning as finding a representation h such that with probability $\geq 1-\delta$ over the data D and the randomness of the algorithm itself, satisfies $Pr_{\overline{x}\in D}[h(\overline{x}) = c(\overline{x})] \geq 1-\epsilon$, where c was some representation in the class $\mathscr{C}$.

We say a PAC-Learning algorithm is *proper* if the algorithm returns a representation of $h \in \mathscr{C}$. Otherwise, it is *improper*.

As such, Elimination is proper.

In the next couple of lectures, we see a class of representation for which there exists an improper learning algorithm, but for which we don't believe a proper learning algorithm exists.

3 Proper learning solves Consis

Lemma If there exists an efficient and proper PAC-Learning algorithm for a class $\mathscr{C}$, then there is a randomized polynomial-time algorithm for solving the following NP-Search Problem:

$$Consis_{\mathscr{C}}^{B(n)} = \{([(\overline{x}^{(1)}, b^{(1)}), ..., (\overline{x}^{(m)}, b^{(m)})], c) c \in \mathscr{C}, |c| \leq B(n), \forall i\ c(\overline{x}^{(i)}) = b^{(i)}\}$$

We will be able to show $Consis_{\mathscr{C}}^{B(n)}$ is NP-Complete for some $\mathscr{C}$.

Proof Consider D, which picks one of the m examples from the list $x^1 ... x^m$ uniformly at random. Any h that is wrong on l of the m examples is correct with probability at most $1 - \frac{l}{m}$. It suffices to guarantee that this is greater than $1 - \frac{1}{m}$, since then $l < 1$. So, we can pass $\epsilon = \frac{1}{2m}$ into the PAC-Learning algorithm, δ as given, and $size(c) = B(n)$. If there exists a $c \in \mathscr{C}$ that is consistent with all of the examples, then the PAC-Learning algorithm is guaranteed to run in time polynomial in n, $\frac{1}{2m}$, $\frac{1}{\delta}$, and $B(n)$ and return $c' \in \mathscr{C}$ such that with probability $\geq 1-\delta$, $Pr[c'(\overline{x} = b)] \geq 1 - \frac{1}{2m}$. Thus, c' must compute all of the labels correctly, so it solves $Consis_{\mathscr{C}}^{B(n)}$. ■

Preview of the next class The class that is not properly learnable is 3-term DNF, which are ORs of three ANDs of literals.

$$T_1 \vee T_2 \vee T_3$$

where each T_i is a conjunction.

We will show $Consis_{(3\text{-}TermDNF)}$ is NP-Hard but there exists an improper learning algorithm for 3-Term DNF that uses 3-CNF representation.