1 Today’s Topics

1. NP-Completeness
2. Representing Computation
3. "Proper” vs. "Improper” Learning

2 NP-Completeness

Since PAC-learning is a question of whether or not efficient algorithms exist for a problem, establishing that a learning problem can’t be solved requires proving that no efficient algorithm can solve the problem.

Reductions will give us a means to prove that problems don’t have efficient algorithms. If, using a hypothetical algorithm for some task as a subroutine, we can obtain an efficient algorithm for some other problem that is known (or believed) to be intractable, then we can conclude that any algorithm for our task can’t be efficient. "NP-Complete” problems will be our first canonical example of such "problems believed to be intractable.”

2.1 NP Search Problems

2.1.1 Definition

A relation problem $R \subseteq X \times Y$ is an NP Search Problem if

1. $R$ is "polynomially balanced" – there is a polynomial $p(n)$ such that for every $(x, y) \in R$, there exists a $y'$ with $|y'| \leq p(|x|)$ and $(x, y') \in R$, where $|y'|$ is its length in bits.
2. $R$ is "efficiently verifiable" – there is a polynomial-time algorithm for computing the Boolean function

$$R(x, y) = \begin{cases} 0, & \text{if } (x, y) \notin R \\ 1, & \text{if } (x, y) \in R \end{cases} \quad (1)$$

2.1.2 Example

For example, if $C$ is a class of efficiently evalutatable representations and $B(n)$ is a polynomial, then the problem

$$Consis_C^{B(n)} = \{ (\bar{x}, c) : c \in C \text{ is consistent with } \bar{x} \text{ and } |c| \leq B(|\bar{x}|) \} \quad (2)$$

is an NP-Search Problem. This is true because the condition that $|c| \leq B(|\bar{x}|)$ satisfies the “polynomially balanced” condition and $C$ being efficiently evalutatable implies the existence of a polynomial time algorithm for testing if $c$ is consistent with $\bar{x}$, so the “efficiently verifiable” condition is satisfied. Specifically, we can test if $c(x^{(i)}) = b^{(i)}$ for all $(x^{(i)}, b^{(i)}) \in \bar{x}$. This runs in polynomial time since we can evaluate $c(x)$ in polynomial time.

For example, we saw that the Elimination algorithm solves $Consis_C^{B(n)}$ for $C =$ the class of conjunctions and $B(n) = O(n)$. 
2.2 NP-Complete Problems

2.2.1 Definition

We will say that a relation problem $R$ is $NP-Complete$ if:

1. $R$ is an NP-search problem
2. for every NP-search problem $S$, there is an algorithm that uses an algorithm for solving $R$ as a subroutine, and assuming that this subroutine runs in polynomial time, solves $S$ in polynomial time.

2.2.2 Discussion

An efficient algorithm for any NP-complete problem gives an efficient algorithm for every NP-search problem. On account of the diverse variety of NP-search problems that have resisted years of intense effort at the design of algorithms, we strongly believe (but don’t yet know) that no algorithm exists for any NP-complete problem that is efficient. The solution to this question is the "P vs. NP" problem. It is relatively straightforward to show that problems are NP-complete: we only need to reduce another (known) NP-complete problem to our problem.

Let $R'$ be an NP-search problem that we want to show is NP-complete, and let $R$ be a known NP-complete problem. Starting from any other NP-search problem $S$:

1. We can obtain an efficient algorithm for $S$ using an efficient algorithm for the known NP-complete problem $R$.
2. Next, we obtain an efficient algorithm for $R$ using the reduction from $R$ to $R'$.

So given an efficient algorithm for $R'$, since polynomial time is closed under composition, we obtain a polynomial-time algorithm for $S$. So $R'$ is also NP-complete.

3 Representing Computation

One way to represent computation is through the use of Boolean Circuits. This is believable since, at some level, real computers are implemented using interconnected logic gates.

3.1 Definition

A Boolean Circuit is given by a directed acyclic graph (DAG) in which:

1. The sources are labeled either by attributes $x_1, ..., x_n$ or constants in $\{0,1\}$.
2. The $k$ sinks of the graph are labeled distinctly by $y_1, ..., y_k$, and each sink has in-degree 1.
3. The internal nodes are labeled by $\land$, $\lor$, or $\lnot$, where $\land$ and $\lor$ nodes have in-degree 2 and $\lnot$ nodes have in-degree 1.

The Boolean circuit computes a function $f : \{0,1\}^n \rightarrow \{0,1\}^k$ as follows:

1. Label the sources with the corresponding value from the input.
2. Assign Boolean labels to the rest of the nodes as follows: compute the Boolean function on the node's label using the Boolean labels of the nodes with edges to it as inputs.
3. Read off the values of $y_1, ..., y_k$ from the sinks as output.

Note that this algorithm for evaluating the circuit runs in $O(|c|)$ time.
3.2 Example

\[ f(1, 0) = (0, 1) \]

3.3 Proposition

Suppose there is a polynomial-time algorithm that computes \( f = \{ f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{k(n)} \} \). Then there is an algorithm that when given an integer \( n \) as input, runs in polynomial time in \( n \) and outputs a representation of a Boolean circuit that computes \( f \) on inputs of length \( n \), i.e., the circuit \( c_n \) that satisfies \( c_n(x_1, ..., x_n) = f_n(x_1, ..., x_n) \) for all \( x \in \{0, 1\}^n \). This proposition will be useful to us in the next section.

4 CIRCUIT-SAT

Consider the problem \( \text{CIRCUIT-SAT} = \{(c, x) : c \text{ is a Boolean Circuit and } c(x) = 1 \} \).

4.1 Theorem ("Cook - Levin")

\( \text{CIRCUIT-SAT} \) is NP-Complete.

Proof:

1. \( \text{CIRCUIT-SAT} \) is an NP-search problem since any input on which \( c \) will evaluate to 1 is no larger than the circuit itself, i.e. \( |x| \leq |c| \), so the problem is “polynomially balanced.” Our circuit evaluation algorithm runs in \( O(|c|) \) time (which is polynomial time) so the problem is “efficiently verifiable”.

2. Let any NP-search problem \( S \) be given. We can modify \( S \) slightly to only use "witnesses" of length \( p(n) \) for the polynomial \( p \) guaranteed by the polynomial-balance condition. We add some "padding" to \( y \) to give it length exactly \( p(n) \) and strip this padding off, given a padded \( y \) to solve the original \( S \). We now find a circuit to solve the padded version of \( S \) by our proposition and the efficient verifiability condition.

We will complete this proof in the next lecture.