CSE 513T Theory of Artificial Intelligence & Machine Learning January 22, 2015

Lecture 4

Lecturer: Brendan Juba *Scribe: Andrew Deck*

1 From last class

1.1 NP-search definition

$R \subseteq X \times Y$ is an NP-search problem if:

1. R is "polynomially balanced"; there is a polynomial p(n) s.t. for every x s.t. $(x, y) \in R$, there is some (other) y' s.t. $|y'| \leq p(|x|)$
2. *"efficient verification"* $\cdots$ There is a a polynomial-time algorithm for computing

$$R(x, y) = \begin{cases} 1 & (x, y) \subseteq R \\ 0 & otherwise \end{cases}$$

1.2 NP-complete

An NP-search problem is NP-complete if every other NP-search problem S "reduces" to R. In other words, there is an algorithm for S using an algorithm for R as a subroutine that is polynomial time (whenever its subroutine is).

1.3 Proposition

Suppose that there is a polynomial-time algorithm for computing a function $f = \{f_n\{0,1\}^n \to \{0,1\}^k(n)\}$. Then there is an algorithm that, given as input an integer n, runs in polynomial time (in n) and computes a Boolean circuit C_n s.t. $\forall x \in \{0,1\}^n, f_n(x) = C_n(x)$.

In other words, a boolean function mapping sequences of bits onto sequences of bits can be converted into circuits in polynomial time.

Proving this requires a more rigorous definition of an "algorithm" (which includes defining machine model / computational model), so we won't do that in this class.

2 CIRCUIT-SAT

Given a circuit, find an inputs that outputs 1. // TODO – insert figure 1

Theorem 1 Cook-Levin Theorem *CIRCUIT-SATis NP-complete*

Proof Let an arbitrary search problem S be given.

```
S(x)
    generate the circuit C_{|x|}(x, y) = 1 : (x, y) ∈ S ∧ |y| ≤ p(|x|)
    replace the x (input) variables of C_{|x} with my values for x as constants
        call this C|_x s.t. only y variables are inputs; y = CIRCUIT − SAT(C|_x)
    return y s.t. (x, y) ∈ S
```

■

2.1 Simpler subproblem: 3SAT

Show that the 3SAT problem is NP-complete where 3SAT is a 3 CNF (conjunctive normal form), which is an AND of at most 3 ORs of literals (i.e. $(x_1 \vee \neg x_3 \vee x_5) \wedge x_7$ is a 3 CNF because it is of the form $L1 \wedge L2 \wedge L3$ where each of $L1 \cdots L3$ is simply a disjunction of zero or more literals).

$$3SAT = (\phi, x), \phi isa 3CNF where \phi(x) = 1$$

Show how to solve CIRCUIT-SATfor 3SAT ("reduction"). Since every S in NP can be soled using a subroutine CIRCUIT-SAT, by composing the algorithm we get an algorithm for solving (the arbitrary) S in NP. Using a subroutine for 3SAT implies 3SAT is also NP-complete.

```
CIRCUIT-SAT( C )
    create a 3CNF by creaeting a variable for each wire of C
    add the gate constraints for each gate to create φ_C and add clause
        "y" to φ_C .
    (X = inputs, w = wires, y = 1) = 3SAT(φ_C
output x : C(X) = 1 // where C(x) is a set assignment to C, since x directly
                    // determines the values on w, which are chosen to make y = 1.
```

To determine the constraints imposed on the wire variables by the output variables, write the relationship between the wire variables and output variables as an implication, then flip the implication (because it is biconditional) and translate to basic boolean operators (via the equivalence $A \implies B \equiv \neg A \vee B$).

3 Learning

Definition 2 Efficiently PAC learnable *We say that $C = \bigcup_{i=1}^{n} C_i$ is efficiently PAC learnable if there is an algorithm that for every distribution D on X_n when given n, $\epsilon \in (0, \frac{1}{2})$, $\delta \in (0, \frac{1}{2})$, $size(c)$, and access to examples drawn from D and labeled by c independently, runs in polynomial time in those parameters, and with probability $1 - \delta$ produces as output a representation h s.t.*

$$Pr_{x \in D}[c(x) \neq h(x)] < \epsilon$$

Definition 3 Proper / Improper Learning Algorithm *An algorithm is a proper learning algorithm if it learns a class C by producing a representation for class C (e.g. our elimination algorithm learned conjunctions by producing conjunctions, and learned disjunctions by producing disjunctions).*

Conversely, an improper learning algorithm learns a class C by producing a more expressive representation than is necessary for class C.

In other words, proper learning means:

$$C_n \equiv H_n$$

And improper learning means:

$$C_n \subset H_n$$

Improper learning is useful because *proper* learning of 3-term DNF (also known as "sum-of-products form"; just 3-term CNF with ORs and ANDs flipped) formulae is an NP-complete problem, but there is an algorithm that learns 3-term DNF formulae by producing a more expressive class.

3.1 Improper Learning Algorithm for 3-term DNFs

Lemma 4 if there is a proper PAC learning algorithm for a class C then there is a randomized polynomial time algorithm for solving the following NP search problem:

$$Consis_C^{B(n)} = \{([(x^1, b^1), \cdots, (x^n, b^n)], c) : \forall i \, c(x^i) = b^i, |c| \leq B(n)\}, c \in C$$

"find $c \in C$ of length $\leq B(n)$ that is consistent"

For any desired confidence $\delta \in (0, \frac{1}{2})$ we run the PAC-learning algorithm with inputs $(n, \frac{1}{2m}, \delta, B(n))$, and given access to uniform distribution on input x's. Notice, whenever some $c' \in C$ is consistent with the data, $b^i = c'(x^i)$ so *the distribution we simulate is exactly labeled x's from $D = x^i$ w.p. (with probability) $\frac{1}{n}$.*

It's safe to assume $\exists c', |c'| \leq B(n)$ s.t. $c'(x^i) = b^i$.

Therefore we have (with D as stated above):

$$h = L^D(n, \frac{1}{2m}, \delta, B(n))$$

Theorem 5 *$Consis^{3n}_{3-termDNF}$ is NP-complete. Thus if there were a proper PAC-learning algorithm for 3-term DNF, by the lemma there exists a randomized algorithm for $Consis^{3n}_{3-termDNF}$ for every NP search problem. So the PAC-learning algorithm runs in polynomial time and w.p. $1 - \delta$ produces h s.t. $Pr_{x \in D}[h(x) = c(x)] \geq 1 - \epsilon = 1 - \frac{1}{2m}$.*

Notice, each $x \in D$ is drawn w.p. $\frac{1}{m}$, so if h is wrong *on any $x \in D$, $Pr_{x \in D}[h(x) = c(x)] \leq 1 - \frac{1}{m} < 1 - \frac{1}{2m}$, which violates the PAC-learning guarantee. Therefore, w.p. $1 - \delta$, the algorithm returns a consistent h.*