Lecture 6: Computational Complexity of Learning—Proper vs Improper Learning
For any family of hypothesis classes consider:

- **FINDCONS**\(\mathcal{H}\): Given a sample \(S\), either return \(h \in \mathcal{H}_n\) consistent with the sample (i.e. s.t. \(L_S(h) = 0\)), or declare that none exists.

- Decision problem \(\text{CONS}\(\mathcal{H}\)(S) = 1\) iff \(\exists h \in \mathcal{H}_n\) s.t. \(L_S(h) = 0\).

**Theorem:** If \(\mathcal{H}_n\) over \(\mathcal{X}_n = \mathbb{R}^n\) is efficiently properly PAC learnable, then

- \(\text{VCdim}(\mathcal{H}_n) \leq \text{poly}(n)\), and
- \(\text{CONS}\(\mathcal{H}\) \in \text{RP}\)

**3 – TERM – DNF**\(\mathcal{H}_n\) = \(\{T_1 \lor T_2 \lor T_3 \mid T_i \in \text{CONJ}_n\}\)

E.g. \(h(x) = (x_1 \land \overline{x[7]} \land x[23] \land x[75]) \lor (\overline{x[2]} \land x[3]) \lor (x[5] \land x[6] \land x[7])\)
Hardness of 3-TERM-DNF

- Claim: $CON S_{3-TERM-DNF}$ is NP-hard

- Proof: Reduction from graph 3-colorability

$3COLOR = \{ \text{undirected graph } G(V,E) \mid \exists c: V \to \{1,2,3\} \ \forall (u-v) \in E c(u) \neq c(v) \}$

- Map $G(V,E) \mapsto S_G$ s.t. $G \in 3COLOR \iff S_G \in CON S_{3-T-DNF}$

- $S_G = \{ (x_i = (1,1,1, ..., 1,0,1, ..., 1,1), y = 1) \mid i = 1..|V| \}$

  $\cup \{ (x_{ij} = (1,1, ... 1,0,1,1 ..., 1,0,1,1 ..., 1,1), y = 0 \mid (i - j) \in E \}$

- Claim: $G \in 3COLOR \Rightarrow S_G \in CON S_{3-T-DNF}$

  - $S_G$ satisfied by $T = T_1 \lor T_2 \lor T_3$ where $T_k = \land_{c(i) \neq k} x[i]$ where $c(i) = \min k \text{ s.t. } T_k \text{ satisfies } x_i$
Hardness of Learning 3-TERM-DNF

- **Conclusion**: If $NP \neq RP$, then $3-TERM-DNF$ is not efficiently properly PAC learnable

- **Proof**: $3-TERM-DNF$ efficiently properly PAC learnable

  \[ \downarrow \]

  $CONST_{3-TERM-DNF} \in RP$

  \[ \downarrow \]

  $3COLOR \in RP$

  \[ \downarrow \]

  $RP = NP$

- **Theorem**: Assuming $NP \neq RP$, $\mathcal{H}_n$ over $\mathcal{X}_n = \{0,1\}^n$ of $\mathcal{X}_n = \mathbb{R}^n$ is efficiently **properly** PAC learnable iff:
  
  - $VCdim(\mathcal{H}_n) \leq poly(n)$, and 
  
  - There is a poly-time algorithm for $CONS_{\mathcal{H}}$ (polynomial in size of input)

- **What about improper learning?**
Hardness of CONSISTENT

- Axis-aligned rectangles in $n$ dimensions
- Halfspaces in $n$ dimensions
- Conjunctions on $n$ variables

- 3-term DNF’s
- DNF formulas of size poly(n)
- Generic logical formulas of size poly(n)
- Python programs of size at most poly(n)
- Decision trees of size at most poly(n)
- Neural nets with at most poly(n) units
- Functions computable in poly(n) time

CONSISTENT: Poly-time

What does this imply?

CONSISTENT: NP-Hard

What does this imply?
Another Look at 3-TERM-DNF

• $3TERM - DNF_n = \{T_1 \lor T_2 \lor T_3 \mid T_i \in CONJ_n\}$

E.g. $h(x) = (x[1] \land \overline{x}[7] \land x[23]) \lor (\overline{x}[2] \land x[3]) \lor (x[6] \land x[7])$

Can also write: $h(x) = (x[1] \lor \overline{x}[2] \lor x[6]) \land (x[1] \lor \overline{x}[2] \lor x[7]) \land (x[1] \lor x[3] \lor x[6]) \land (x[1] \lor x[3] \lor x[7]) \land (\overline{x}[7] \lor x[2] \lor x[6]) \land (\overline{x}[7] \lor x[2] \lor x[7]) \land (\overline{x}[7] \lor x[3] \lor x[6]) \land (\overline{x}[7] \lor x[3] \lor x[7]) \land (x[23] \lor x[2] \lor x[6]) \land (x[23] \lor x[2] \lor x[7]) \land (x[23] \lor x[3] \lor x[6]) \land (x[23] \lor x[3] \lor x[7])$

• Claim: $3TERM - DNF_n \subset 3CNF_n$

$3CNF_n = \{ \land_{i=1}^{r} T_i \mid T_i = \lor \text{of three literals} \}$

• But $3CNF_n \subset CONJ\{\text{disjunctions of three literals}\}$
  • Can be efficiently learned using $FINDCONS_{CONJ}$, treating each $(2n)^3$ 3-disjunction as a single variable
  • I.e. map $x \mapsto (x[1] \lor x[2] \lor x[3], x[1] \lor x[2] \lor x[4], ..., x[1] \lor x[2] \lor$
Efficiently Learning 3-TERM-DNF

- Sample complexity of learning 3-TERM-DNF using $FINDCONS_{3CNF}$: $O\left(\frac{n^3}{\epsilon}\right)$
- Compare with cardinality bound: $O\left(\frac{n}{\epsilon}\right)$
- Why the gap?

- Relax statistically easy but computationally hard class to larger class that’s statistically harder but computationally easier:

<table>
<thead>
<tr>
<th></th>
<th>$CONS_{3-T-DNF}$</th>
<th>$CONS_{3CNF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample complexity</td>
<td>$O\left(\frac{n}{\epsilon}\right)$</td>
<td>$O\left(\frac{n^3}{\epsilon}\right)$</td>
</tr>
<tr>
<td>Runtime</td>
<td>$2^O(n)/\epsilon$</td>
<td>$O\left(\frac{n^6}{\epsilon}\right)$</td>
</tr>
</tbody>
</table>
Hardness of CONSISTENT

- Axis-aligned rectangles in $n$ dimensions
- Halfspaces in $n$ dimensions
- Conjunctions on $n$ variables
- 3-term DNF’s
- DNF formulas of size poly($n$)
- Generic logical formulas of size poly($n$)
- Decision trees of size at most poly($n$)
- Neural nets with at most poly($n$) units
- Functions computable in poly($n$) time
- *Python programs of size at most poly($n$)*

**CONSISTENT:**

- **Poly-time** Efficiently Properly Learnable
- **NP-Hard** Not Efficiently Properly Learnable
- Improperly Learnable?
Proper vs Improper Learning

- 3-TERM-DNF is not efficiently properly learnable, but is efficiently (improperly) learnable.
- Are there classes that are not efficiently learnable, even improperly?
- What about the class of all functions computable in time $\text{poly}(n)$?
  - Universal class: any $\mathcal{H}_n$ where the predictors are poly-time computable is $\mathcal{H}_n \subseteq \text{TIME}(\text{poly}(n))$
  - $VCDim = O(\text{poly}(n))$
  - CONS is NP-Complete:
    - If $P = NP$ we could actually learn this universal class!
    - If $RP \neq NP$, we can’t efficiently properly learn it
    - But can we improperly learn it?
- How do we show that a class cannot be learned, no matter what hypothesis we are allowed to output?
Learning and Crypto

- How to attack the cryptosystem:
  - Usually assume you know the form of $f$ (but not $K$)
    e.g. you know target is using a GSM phone, or ssh, or file format specifies encryption, or you captured an Enigma machine
  - Collect messages for which you know the plaintext, i.e. pairs $(s_i, c_i)$
    ... or have partial or statistical information on the plaintext $s_i$, e.g. $s_i$ is an English word, and so $\Pr[s_i[1] = "t"] = 0.17$
  - Search for a key $K$ s.t. $c_i = f_K(s_i)$

- Information Theoretically:
  - Need $m = \frac{\log|\mathcal{K}|}{n-H(s)}$ message pairs
  - Only way to be secure:
    One Time Pad, i.e. $\mathcal{K} = \{0,1\}^{m \cdot n}$

Claude Shannon 1916-2001
Learning and Crypto

- Reducing Code Breaking to Machine Learning:
  - Hypothesis class \( \{ f_K^{-1} \mid K \in \mathcal{K} \} \) over domain \( \mathcal{X} = \{ \text{encrypted messages} c \} \) and targets \( \{ \text{plaintext messages} s \} \)
  - Training set \( \{(c_i, s_i)\}_{i=1}^m \), possibly noisy \( s_i \)
  - Learn decoder \( c \mapsto s \)

- Proper learning: find decoder of the form \( f_K^{-1} \)
- But really, finding any decoder that decodes most messages is fine \( \Rightarrow \) Improper learning
- Statistically, enough to have \( m \propto \log|\mathcal{H}| \leq \log|\mathcal{K}| \) messages
- So it better be computationally hard...
  - Learning is computationally easy \( \Rightarrow \) Breaking code is easy \( \Rightarrow \) No crypto
  - Crypto is possible \( \Rightarrow \) learning is computationally hard
Discrete Cube Root Cryptosystem

- Choose primes $p, q$ s.t. $3 \nmid (p - 1)(q - 1)$
- Public (encryption) key: $K = p \cdot q \approx 2^n$
- Private (decryption) key: $D = \frac{1}{3} \mod (p - 1)(q - 1)$
- $f_K(s) = s^3 \mod K$
  - easy to compute from $K$ and $s$ in time $O(n^2)$
- $f_K^{-1}(c) = c^D \mod K$
  Proof: $3D = a(p - 1)(q - 1) + 1$ for some integer $a$ and so mod $K$ we have $(c^D)^3 = c^{3D} = (c^{p-1})^a(c^{q-1})^a c = 1^a 1^a c = c$
  - easy to compute from $D$ and $c$ in time $O(n^3)$
- Hard to compute from $K$ and $c$
- Hard in what sense?
  - Worst case: no poly time alg that works for every $c$
    maybe only hard on crazy $c$, but we can decrypt most messages
  - We would like: no poly time alg that works for any $c$
    impossible—always return 3487, you’ll be right on some $c$
  - Enough: no poly time alg that works for random $c$
    can make $c$ random by sending $f_K(s \oplus r), r$ for random $r$
Cryptographic Assumption

1. There exists a poly-time algorithm $ENC(K, s) = f_K(s) = s^3 \mod K$

2. For every $K \in \mathcal{K}$, there exists $D(K)$ and a poly-time algorithm $DEC(D(K), c) = f_K^{-1}(c) = s^D \mod K$, i.e. s.t. $\forall s \in \mathcal{S} DEC(D(K), f_K(s)) = s$

3. There is no poly-time (randomized) algorithm $BREAK(K, c)$ s.t. $\forall K \in \mathcal{K}$
   $$\Pr_{s \sim \text{Unif}(\mathcal{S}), \text{randomness in } B}[BREAK(K, f_K(s)) = s] \geq \frac{1}{\text{poly}(n)}$$

$\mathcal{K}, \mathcal{S} \subseteq \{0,1\}^n$, i.e. size of inputs is $O(n)$

For discrete cube root:
- $\mathcal{K} = \{ K = (p - 1)(q - 1) \mid p, q \text{ prime, } 2^{n-2} \leq K < 2^n \}$
- $\mathcal{S} = \{0,1\}^{n-2} = \{s \mid 0 \leq s < 2^{n-2} \}$

- For a useable public-key cryptosystem also need:
  - A poly($n$) algorithm for generating $(K, D(K))$ -- we won’t use this
  - $\Pr_{K,s}[BREAK(K, f_k(s)) = s] \geq \frac{1}{\text{poly}(n)}$ rather then $\forall K \ Pr[s] \leq \frac{1}{\text{poly}(n)}$
    ($\forall K$ enough for us)
Hardness of Learning
Discrete Cube Root

$$\mathcal{H}_n = \{ (c, i) \mapsto f_K^{-1}(c)[i] \mid K \in \mathcal{K}, i = 1..n - 2 \}$$

- VCDim($\mathcal{H}_n$) $\leq \log|\mathcal{H}_n| \leq \log|\{0,1\}^n| = n$
- All predictors in $\mathcal{H}_n$ are computable in time $O(n^3)$
- **Claim**: Subject to discrete cube root assumption, $\mathcal{H}_n$ is not efficiently PAC learnable.
- Proof: Given poly-time learning algorithm $A$, construct $BREAK(K, c)$:

  For i=1..n, $h_i \leftarrow A\left(D_i, \epsilon = \frac{1}{4n}, \delta = \frac{1}{4n}\right)$
  
  $D_i(x, y)$: $a \sim Unif(\{0,1\}^{n-2})$
  $x = (f_K(a), i)$, $y = a[i]$

  Return $[h_1(c), h_2(c), ..., h_{n-2}(c)]$

W.p. $\geq 1 - \frac{n-2}{4n} > \frac{3}{4}$ (over randomness in $BREAK$) learning “worked” for all $i$, and in that case, the probability to err on some bit is $\leq \frac{n-2}{4n} < \frac{1}{4}$ (over the choice of $c$), and so overall w.p.$>1/2$, $BREAK(K, c) = f_K^{-1}(c)$. 
We identified a family of functions that is provably not efficiently PAC-learnable
  - Despite having polynomial VC-dimension
  - Despite containing only easily computable functions
  - Subject to an assumption about security of a cryptosystem

Since $\mathcal{H}_n \subseteq \text{TIME}(O(n^3))$, this implies poly-time computable functions are not efficiently PAC-learnable

In fact, if there is any secure public-key crypto-system, then poly-time computable functions are not efficiently PAC-learnable

Even stronger: if we can efficiently learn poly-time computable functions then no cipher is secure to known-plaintext attacks
Reducing Breaking an unknown code to learning poly-time functions:

- Hypothesis class \( \{ \text{poly-time computable } h: \mathcal{X} \rightarrow \mathcal{Y} \} \) over domain \( \mathcal{X} = \{ \text{encrypted messages } c \} \) and targets \( \{ \text{plaintext messages } s \} \)
- Training set \( \{(c_i, s_i)\}_{i=1..m} \), possibly noisy \( s_i \)
- Learn decoder \( c \mapsto s \)

Conclusion: if we can efficiently learn poly-time computable functions, we can break any cipher with a known-plaintext attack

\( \Rightarrow \) If there exists any secure cipher system, then poly-time functions are not efficiently PAC learnable
Hardness of Learning via Crypto

• Public-key crypto is possible (one way functions exist)
  ➤ hard to learn poly-time functions

• Hardness of Discrete Cube Root
  ➤ hard to learn log(n)-depth logic circuits
  ➤ hard to learn log(n)-depth poly-size neural networks

• Hardness of breaking RSA
  ➤ hard to learn poly-length logical formulas
  ➤ hard to learn poly-size automata
    ➤ hard to learn push-down automata, ie regexps
  ➤ for some depth d, hard to learn poly-size depth-d threshold circuits
    (output of unit is one iff number of input units that are one is greater than threshold)

• Hardness of lattice-shortest-vector based cryptography
  ➤ hard to learn intersection of $n^r$ halfspaces (for any $r > 0$)