CSCI4230 Computational Learning Theory — Spring 2021

Lecturer: Siu On Chan — *Based on Varun Kanade's notes*

Notes 24: Inherent hardness of learning

Notes11 showed that $\mathcal{C} = \{\text{3-term DNFs}\}$ is hard to be PAC-learned properly (assuming $\mathrm{NP} \neq \mathrm{RP}$)
But $\mathcal{C}$ can be efficiently PAC learned improperly with hypothesis class $\mathcal{H} = \{\text{3-CNFs}\}$
Question: Is there $\mathcal{C}$ that is hard to be PAC-learned improperly, regardless of the hypothesis class?
Answer: Yes, under **cryptographic** assumptions

---

1. Discrete Cube Root

Let p and q be two large primes requiring roughly the same number of bits to represent
e.g. both n bits (so each prime is between 2^n and $2^{n+1}-1$ in magnitude)
Consider all integers modulo pq (i.e. between 0 and $pq-1$), denoted by $\mathbb{Z}_{pq} = \{x \in \mathbb{Z} \mid 0 \leqslant x < pq\}$
Among numbers in $\mathbb{Z}_{pq}$, consider those that are coprime to pq
Recall: x coprime to pq $\iff$ x and pq share no common factors other than 1
$\iff$ greatest common divisor (gcd) of x and pq is 1
Denote the set of such numbers by $\mathbb{Z}_{pq}^\times = \{x \in \mathbb{Z}_{pq} \mid \gcd(x, pq) = 1\}$
Easy to check that $|\mathbb{Z}_{pq}^\times| = (p-1)(q-1)$ (x has to be both coprime to p and coprime to q)
This is called Euler phi function φ of pq, so $\varphi(pq) = |\mathbb{Z}_{pq}^\times| = (p-1)(q-1)$
$\mathbb{Z}_{pq}^\times$ forms an (abelian) group under multiplication modulo pq
In particular, if x and y are both coprime to pq, then so is $xy \bmod pq$

Now assume 3 does not divide $|\mathbb{Z}_{pq}^\times| = \varphi(pq) = (p-1)(q-1)$
This happens if and only if both p and q are of the form $3k+2$
Then $f_{pq} : \mathbb{Z}_{pq}^\times \to \mathbb{Z}_{pq}^\times$ given by $f_{pq}(y) = y^3$ is bijective
Reason: $\varphi(pq)$ coprime to 3
$\iff$ $3d = \varphi(pq)b + 1$ for some integers d, b (due to extended Euclidean algorithm)
$\implies$ $(f_{pq}(y))^d \equiv y^{3d} \equiv y^{\varphi(pq)b+1} \equiv y \pmod{pq}$ (using Euler's theorem)

Given $N = pq$ and $y \in \mathbb{Z}_{pq}^\times$, it's easy to compute $f_N(y) = y^3$ (its cube mod N)
Given $N = pq$ and $x \in \mathbb{Z}_{pq}^\times$, seems hard to find $y = f_{pq}^{-1}(x)$ (cube root of x mod N)
Discrete Cube Root problem
Input: $N = pq$ and $x \in \mathbb{Z}_N^\times$, where p and q are unknown n-bit primes and $\gcd(3, \varphi(pq)) = 1$
Output: y such that $y^3 \equiv x \pmod N$

Discrete Cube Root Assumption: Any randomized polynomial time algorithm A
When given inputs N and x as above
Where unknown primes p and q are random n-bit primes of the form $3k+2$
And where $x \in \mathbb{Z}_N^\times$ is uniformly random
A manages to find the cube root of x with only negligible probability
"Polynomial time" means polynomial in n (not N), since numbers are represented as n-bits
More precisely, negligible probability means probability $1/n^{\omega(1)}$
i.e. decays faster than any inverse polynomial

---

2. RSA

Discrete Cube Root problem is used in RSA cryptography (when the public key is 3)
Discrete Cube Root Assumption (not Factoring Assumption) is why RSA is secure
When public key is 3: any one can take cube mod N to encrypt a messages
d above is the private key: take cube root (equivalent, d-th power) to decrypt an encrypted message

---

3. Cryptographic hardness of learning

Can formulate finding the cube root of $x \in \mathbb{Z}_N^\times$ as a learning problem (given N)

Input: samples $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathbb{Z}_N^\times$ and $y_i^3 \equiv x_i \pmod N$
Output: hypothesis $h : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ such that $\mathbb{P}_{x \sim \mathbb{Z}_N^*}[h(x) \neq f_N^{-1}(x)] \leqslant \varepsilon$
Again, we require the learning algorithm B to output such a hypothesis h with prob. $\geqslant 1 - \delta$
If Discrete Cube Root Assumption holds, then this learning problem has no efficient algorithm
We can turn learning algorithm B into algorithm A breaking the DCR assumption
By sampling samples (x_i, y_i) by ourselves

Above learning problem is not a usual PAC learning problem (not binary classification)
But this minor technical issue can be easily worked around:
The output $y = f_N(x)^{-1}$ consists of $2n$ bits
Define $2n$ functions $f_{N,1}^{-1}, \ldots, f_{N,2n}^{-1} : \mathbb{Z}_N^\times \to \{0, 1\}$ encoding the $2n$ bits of f_N^{-1}
If for each $1 \leqslant i \leqslant 2n$, $f_{N_i}^{-1}$ can be PAC learned to accuracy $\varepsilon/2n$
We will be able to reconstruct f_N^{-1} to accuracy ε
So at least one of the PAC learning problems for $f_{N,1}^{-1}, \ldots, f_{N,2n}^{-1}$ must be hard