Algorithms for Learning in the Presence of Errors

Report by

Niyati Rao - 1020211512

Pranit Sinha- 1020211818

Under the Supervision of:

Professor Mahavir Jhawar

Department of Computer Science, Ashoka University

Submitted on:

8th December, 2024

Acknowledgement

We express our sincere gratitude to our supervisor, **Mahavir Jhawar**, for their guidance, unwavering support, and insights throughout the duration of this work. Their expertise and meticulous attention to detail have been critical in shaping the direction and quality of this research.

We are thankful to the Computer Science Department at Ashoka University, whose facilities, resources, and academic environment have provided an ideal platform for the completion of this project. We extend our appreciation to the esteemed faculty and staff for their assistance and encouragement during this endeavor.

Additionally, We acknowledge the collaborative spirit and constructive feedback from our colleagues and peers, which have greatly enriched the research process.

Finally, We are deeply indebted to our family and close associates for their patience, encouragement, and moral support, which have been a constant source of motivation throughout this journey.

Certificate by the Supervisor

This is to certify that the work presented in this report has been successfully defended and approved by the undersigned supervisor.

Supervisor's Signature: ____________________

Date: ____________________

Contents

Chapter 1

Introduction

1.1 Parity Problem

The **length-k parity problem** refers to the challenge of learning a parity function defined over the binary space $\{0,1\}^k$. Here, a parity function is a boolean function that takes a binary vector of length k as input and returns the parity (even or odd) of the number of 1s in the vector. Specifically, if the input vector has an even number of 1s, the function outputs 0; otherwise, it outputs 1. The task is made more difficult by the presence of random classification noise at rate η, meaning that each output has a probability η of being flipped.

Our Problem Setup: Let $\mathbf{x} \in \{0,1\}^k$ be the secret binary vector that we want to learn. Suppose we have n observations, represented by an $k \times n$ binary matrix $\mathbf{A}$, where each row $\mathbf{a}_i$ is a random binary vector of length n. Let $\tilde{\mathbf{y}} \in \{0,1\}^n$ be the vector of observed outputs, with each entry defined as:

$$\tilde{y}_i = \mathbf{x} \cdot \mathbf{a}_i \oplus \eta_i$$

where $\mathbf{x} \cdot \mathbf{a}_i$ denotes the dot product modulo 2, and $\eta_i \in \{0,1\}$ is a noise term for each observation. We can rewrite this system in matrix form:

$$\tilde{\mathbf{y}} = \mathbf{x}\mathbf{A} \oplus \eta$$

where:

$$\tilde{\mathbf{y}} = \begin{bmatrix} \tilde{y}_1 & \tilde{y}_2 & \cdots & \tilde{y}_n \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_k \end{bmatrix}, \quad \eta = \begin{bmatrix} \eta_1 & \eta_2 & \cdots & \eta_n \end{bmatrix}$$

and

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2k} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & \cdots & a_{kn} \end{bmatrix}$$

And a single instance$(a_i, \tilde{y})$ is generated as follows:

$$\begin{bmatrix} \cdots & \tilde{y}_i & \cdots \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_k \end{bmatrix} \begin{bmatrix} \cdots & \cdot & a_{1i} & \cdots & \cdot \\ \cdots & \cdot & a_{2i} & \cdots & \cdot \\ \cdots & \cdot & \cdot & \cdots & \cdot \\ \cdots & \cdot & a_{ki} & \cdots & \cdot \end{bmatrix} \oplus \{ \begin{bmatrix} \cdot & \cdot & \eta_i & \cdot & \cdot \end{bmatrix}$$

Goal: Given $\mathbf{A}$ and $\tilde{\mathbf{y}}$, our goal is to determine $\mathbf{x}$ despite the presence of noise η. When we probe the oracle the oracle creates one such instance of the problem and provides us with$\mathbf{A}$ and $\tilde{\mathbf{y}}$ and we know the value on $\eta < 1/2$.

Brute Force Algorithm A brute force Algorithm would draw O(n) labeled examples and consider all possible parity functions, ie2^n parity functions to find the one with least empirical error. In contrast the BKW Algorithm runs in $2^{O(n/\log n)}$ time and requires $(2^{O(n/\log n)}$ labeled examples.

1.2 The BKW algorithm

1.2.1 Main Idea of the Algorithm

Learning Parity Functions Without Noise

In the absence of noise, parity functions can be learned by finding a **basis** of previously-seen examples. Once a basis is found, the label of any new example is just the sum of the labels of the basis vectors (mod 2) that on summation form the example.**F**urther Gaussian elimination can be used to find the target function from the basis of examples. **sum (mod 2) of other examples, its label will be the sum (mod 2) of those examples' labels.**

1.2.2 In the Presence of Noise

When noise is introduced (e.g., with a noise rate of 1/4), then the sum of s labels has noise rate $\frac{1}{2} - \left(\frac{1}{2}\right)^{s+1}$.[Appendix A.2] This means to be able to add correctly with probability $\frac{1}{2} + \frac{1}{\text{poly}(n)}$ (just better than half) we can add together only $O(\log n)$ examples.

The method of summing examples to deduce labels becomes unreliable. Sum of multiple noisy labels leads to **higher noise rates**, to handle noise, only a small number of training examples should be summed to maintain a correct label with high probability.

Considering a particular case: When the parity function depends on the first $k = \log n \log\log n$ bits of the input, instead of relying on Gaussian elimination, which requires summing k examples, the BKW algorithm writes the new test example as a sum of $\frac{k}{\log k} = \frac{\log n \log\log n}{\log n} = O(\log n)$ examples.

This smaller set exists because if there are $poly(n)$ training examples, and assuming each is randomly chosen, for $k = \log n \log\log n$, a small subset of training examples (size $O(\log\log n)$) can be found that sums to any new test example. This is possible because there are many subsets of the required size, $n^{O(\log\log n)} >> 2^k$, that are pairwise independent, providing sufficient variety for the algorithm.

This algorithm does not find the smallest subset possible (which would require solving the *subset sum* problem exactly, effectively allowing an algorithm to to learn class of parity functions that depend on the first $O(\log^2 n)$ bits of input in time $2^{O(\sqrt{n})}$), it is still a significant improvement over Gaussian elimination. Goal of the Algorithm is to recover y in $poly(n)$, for the case$k = O(\log n)$ is trivially so as we know that:

$$a^{\log_a b} = b$$

Applying this property to our expression, we get:

$$2^{\log_2 n} = n$$

Therefore, $2^{\log_2 n}$ simplifies to $O(n)$ time. The algorithm suggested here works for $k = c \log n \log\log n$ for some $c > 0$.

Considering the algorithm with $k \times n$ linear pairs, for an algorithm that runs in polynomial time in our implementation we set values in consideration to the following.

1.2.3 Example Usage:

If we consider $n = 2^8 = 256$,

$\log_2 n$:

$$\log_2 n = \log_2 256 = 8$$

$\log_2 \log_2 n$:

$$\log_2 \log_2 n = \log_2 8 = 3$$

Therefore combined $\log_2 n \cdot \log_2 \log_2 n$:

$$\log_2 n \cdot \log_2 \log_2 n = 8 \times 3 = 24$$

For $n = 256$, IF the total number of bits the algorithm depends on is the first 24 bits then it would run in polynomial time.

Similarly if $n = 2^4 = 16$ or $n = 2^2 = 4$, and the algorithm depends on is the first 8 or 2 bits then it would run in polynomial time.

The implementation of this is documented later in the report.

1.2.4 Results of the Paper

- The **BKW Algorithm** is a polynomial-time algorithm, poly(n), that considers the parity problem of $k \times n$ linear codes in the presence of random noise, where $k = c \log n \log \log n$.
- It provides an efficient, noise-tolerant algorithm for a concept class that is provably not learnable in the Statistical Query (SQ) model, i.e., SQ $\subsetneq$ PAC.
- Any class of functions learnable with k-wise queries $k = O(\log n)$ is also weakly learnable with unary queries

1.3 The Arora-Ge algorithms

In [2], Arora and Ge employ an algorithmic approach rooted in linearization, a technique widely used in cryptanalysis and optimization. The primary idea is to transform nonlinear equations into linear systems by introducing new variables to represent monomials, a strategy akin to Sherali-Adams lift-and-project methods in linear programming. While the concept of linearization is well-established, their contribution lies in its novel application to structured noise settings and its rigorous analysis in these contexts.

To illustrate the utility of their approach, they propose a variant of the Learning Parity with Noise (LPN) problem, referred to as LPN with structured noise. In the classical LPN problem, each oracle call provides a random vector $\mathbf{a} \in \mathrm{GF}(2)^n$ and a noisy bit $b \in \mathrm{GF}(2)$, which is incorrect with probability p. In the structured noise variant, the oracle returns m random vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m \in \mathrm{GF}(2)^n$ and m corresponding bits $b_1, b_2, \ldots, b_m$, with the guarantee that at least $(1-p)m$ of these pairs satisfy $\mathbf{a}_i \cdot \mathbf{u} = b_i$.

This structured noise model ensures the data errors are dispersed rather than concentrated. They note that such models may have implications for machine learning, where standard noise models often yield intractable problems.

Their algorithm for LPN with structured noise operates in time polynomial in n and m, making it efficient when m is constant. Interestingly, the algorithm remains effective even if the oracle adaptively chooses where to introduce noise after observing $\mathbf{a}_i$. The key idea is to treat the structured noise guarantees as a system of degree-m constraints. Linearizing these constraints replaces each monomial $\prod_{i \in S} u_i$ with a new variable y_S, yielding a linear system with $\binom{n}{m}$ variables. We prove that this system admits a unique solution, with the proof detailed in Section 2.

They extend the linearization technique to address the Learning with Errors (LWE) problem under noise rates below $\sqrt{n}$. Their algorithm achieves subexponential runtime in this regime, which sheds light on why existing hardness results—linking LWE to lattice approximation problems—do not apply here. Furthermore, they provide a precise characterization of noise models for which this method is effective.

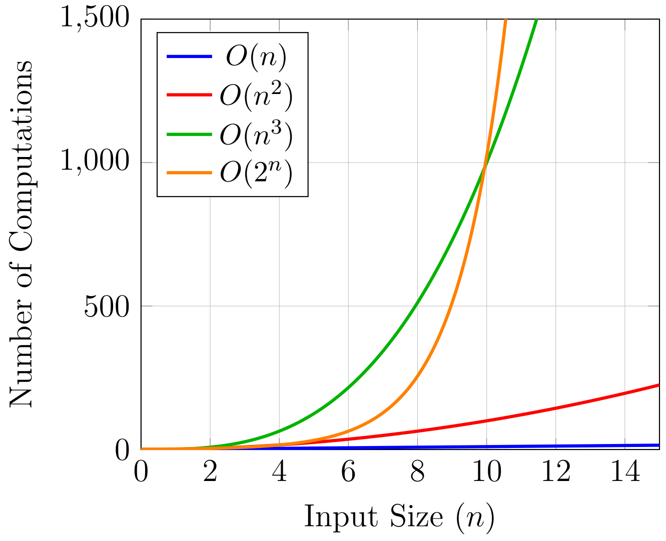
Linearization can be seen as part of a broader trend in mathematical problem-solving: reducing complex problems to linear algebra, particularly Gaussian elimination. In their algorithms, this reduction involves equations with coefficients that are low-degree polynomials in the input data. This approach, which they term reduction to Gaussian elimination with polynomial coefficients, highlights the naturalness of the linearization technique.

Notably, one may observe that every algebraic circuit solving a decision problem can be converted into a form that reduces to Gaussian elimination with polynomial coefficients. This connection underscores the ubiquity of the approach while also hinting at its limitations. Proving lower bounds for this technique remains a challenging task, as it requires progress on fundamental questions in circuit complexity.

Chapter 2

Background and Motivation

Considering solving a system of linear equations we can consider the case of considering a System where there are no error in this case the best known algorithms for finding a solution for such a system of equations is around the order of $O(n^3)$ however this problem becomes a problem with an exponential $O(2^n)$ problem when we consider this system with error introduced in the parity with some probability p.



2.1 Solving Linear Equations without Errors

Problem:

Given a system of m linear equations in n variables:

$$Ax = b$$

where A is an $m \times n$ matrix, x is the n-dimensional variable vector, and b is the m-dimensional output vector, the goal is to solve for x.

Time Complexity:

- **Gaussian Elimination:** Time Complexity: $O(n^3)$ for dense matrices. **LU Decomposition:** $O(n^3)$ for factorization, $O(n^2)$ for solving with precomputed factors.
- **Strassen's Algorithm for Matrix Inversion:** A faster matrix multiplication method with time complexity $O(n^{\log_2(7)})$.

2.2 Solving Parity Problems with Errors

2.2.1 Exponential Set-Up

1. **Problem:** In the problem set-up of the parity problem defined as:

   $$\tilde{\mathbf{y}} = \mathbf{x}\mathbf{A} \oplus \eta$$

   where each y_i is defined as:

   $$\tilde{y_i} = \mathbf{x} \cdot \mathbf{a}_i \oplus \eta_i$$

   at the end for any randomly chosen $m \times n$ block of A the y's must be form a uniform distribution where for each index of y_i the value of 0 and 1 must be equally likely. This is verifiable in our problem instance generation. Below is a documentation of the same.

   - for m = 10, noise rate = 1/3, when run 2000 times, the distribution of each index of y_{0-9} is equally likely to be 1 or 0.

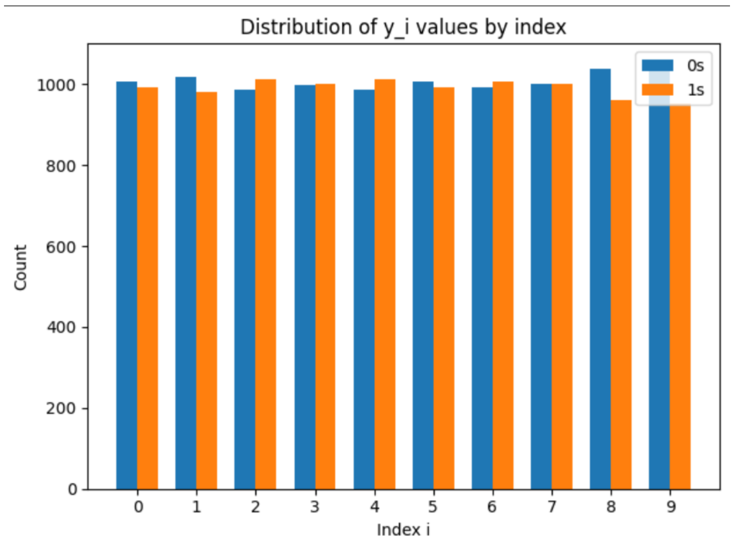


figure 1.Uniform distribution of values of resulting vector y

- considering m= 4, noise rate = 3/4 , run 1000 times, this plot counts the appearance of all 16 possible values of y.

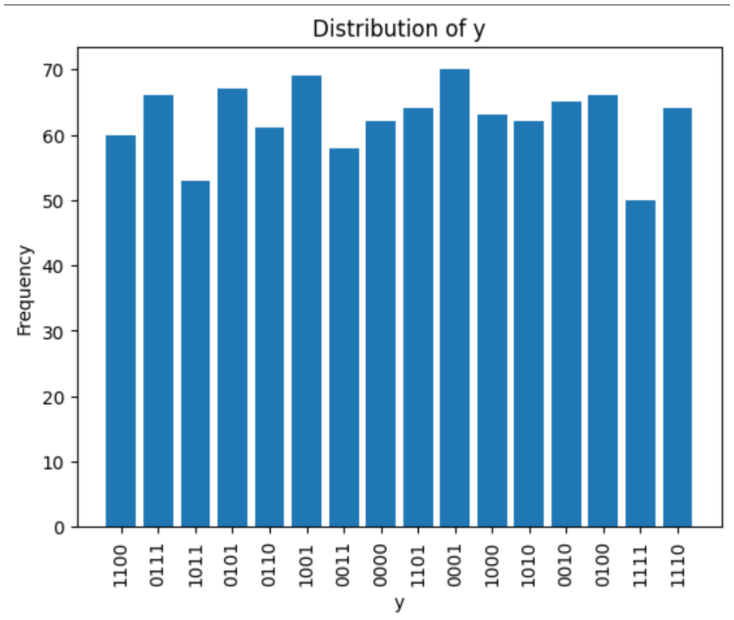


figure 2.Uniform distribution of values of resulting vector y for small m

2. **Exponential Time Solution:**
   Main Idea The goal of the exponential algorithm for solving an LPN

instance is to iterate over all possible secret vectors $u \in \{0,1\}^n$ and test each one. For each candidate u, compute the resulting b' and compare it with the given b. If a match is found, u is the solution. Since the algorithm explores all 2^n possible u's, it is exponential in runtime.

Pseudocode

Input: Matrix $A \in \mathbb{F}_2^{m \times n}$, vector $b \in \mathbb{F}_2^m$, and noise probability p.

Output: Candidate u such that $b \approx (A \cdot u) \mod 2$.

```
(a) Iterate through all possible $u \in \{0,1\}^n$.
(b) For each $u$:
      i. Compute $b' = (A \cdot u) \mod 2$.
     ii. Count how many bits of $b'$ differ from $b$.
    iii. If the number of differing bits is consistent with the noise
         probability $p$, $u$ is a potential solution.
(c) Return the solution.
```

Example Outputs

- Considering a 4 bit secret vector we considered plotting our success rates when each instance had $A(m \times n) = 20 \times 4$ with $p = 0.2$, here is the output for success rates for all 16 possible vectors when 100 trials are run. **Averaging at 80%**

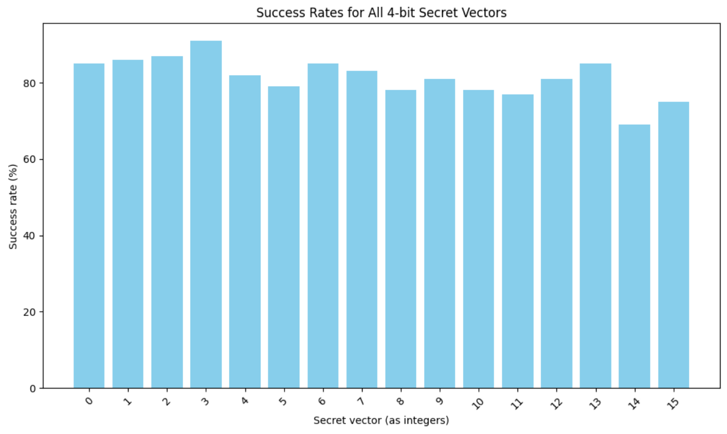


figure 3.Success rates for all possible 4 bit vectors

However **this becomes 100%** when we consider m to be some multiple of2^n

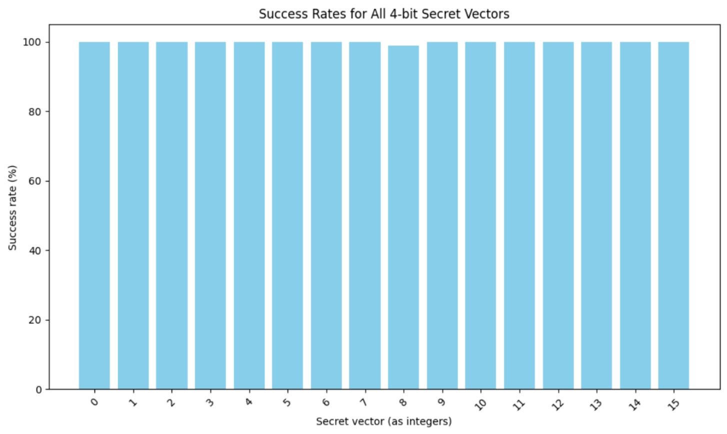


figure 3.Success rate of 100% in exponential time

- We ran simulation for not just 4 bits but for 4, 5, 6, ...16 bits and adjusted the size of n accordingly then on plotting the average success rates over 100 iterations for a 10 random vector of each length we verified an obvious result of the algorithm returning a higher success rate.

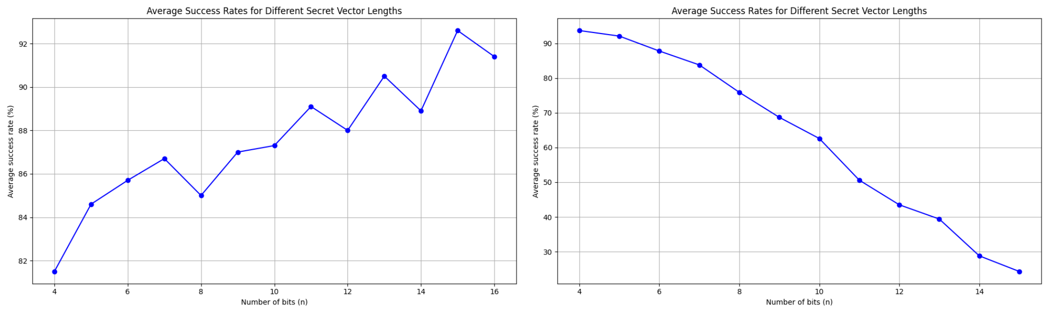


figure 4Success rates increasing length of secret vector with, adjusting (left) vs constant m = 32(right).

The problem lies in the executing LPN instance for large values of n but smaller values of m, that is when the secret key is very large and the number of examples do not account for all possibilities to verify the value of the secret key and further we cannot do Gaussian elimination directly as that would lead the errors carrying over. So an alteration to this technique is considered which is the Blum-Kalai-Wasserman Algorithm.

Subexponential Algorithm - BKW:

The Blum-Kalai-Wasserman (BKW) algorithm is a subexponential algorithm for LPN offers a **Time Complexity:** $2^{O(n/\log n)}$, which is subexponential in n.

2.3 Motivation behind the Project

Through our implementation of the paper in python notebooks we wanted to both check the applicability of the algorithm and verify bounds on randomly created problem instances and work on the algorithm to improve the bounds if possible for certain cases which would be looked into after implementation of BKW algorithm for a larger problem instances.

Chapter 3

Blum-Kalai-Wasserman for LPN

3.1 Literature Review

The details below are from [3].

3.1.1 Preliminaries

- A **concept** is a boolean function over an input space,in the paper $\{0,1\}^n$.
- A **concept class** is a set of concepts.
- The task is to **learn a target concept** in the presence of random classification noise.
- **Noise model**: Each labeled example (x, ℓ) is such that:
  - x is chosen from $\{0,1\}^n$ according to a distribution D,
  - $\ell = c(x)$ with probability $1 - \eta$, and $\ell = 1 - c(x)$ with probability η, where $\eta < \frac{1}{2}$.
- **Learning goal**: Find an ϵ-approximation h of c, such that $\Pr_{x \leftarrow D}[h(x) = c(x)] \geq 1 - \epsilon$. A concept class C is **efficiently learnable** under noise if there exists an algorithm A that, for any $\epsilon > 0$, $\delta > 0$, and $\eta < \frac{1}{2}$,

outputs an ϵ-approximation of $c \in C$ with probability at least $1 - \delta$ in polynomial time.

- A **parity function** c on $\{0,1\}^n$ is defined by a vector $c \in \{0,1\}^n$ such that $c(x) = x \cdot c \pmod 2$.Under the uniform distribution D, parity functions are **pairwise uncorrelated**.

- A parity function **depends on the first k bits** if its nonzero components are within the first k bits, leading to 2^k distinct parity functions.

3.1.2 Learning Parities with Noise

Consider,

Lemma 3

Let $(x_1, \ell_1), \ldots, (x_s, \ell_s)$ be examples labeled by c and corrupted by random noise of rate η. Then $\ell_1 + \cdots + \ell_s$ is the correct value of $(x_1 + \cdots + x_s) \cdot c$ with probability $\frac{1}{2} + \frac{1}{2}(1-2\eta)^s$.

Proof. We use induction on the number of examples s.

Base case: For $s = 1$, the statement holds trivially since the probability that $\ell_1 = x_1 \cdot c$ is $\frac{1}{2} + \frac{1}{2}(1-2\eta)^1 = \frac{1}{2} + \frac{1}{2} - \frac{2\eta}{2}) = 1 - \eta$ and the probability that $\ell_1 \neq x_1 \cdot c$ is η. Thus,

$$\Pr(\ell_1 = x_1 \cdot c) = 1 - \eta = \frac{1}{2} + \frac{1}{2}(1 - 2\eta).$$

Inductive step: Assume the lemma is true for $s-1$ examples. That is,

$$\Pr(\ell_1 + \cdots + \ell_{s-1} = (x_1 + \cdots + x_{s-1}) \cdot c) = \frac{1}{2} + \frac{1}{2}(1-2\eta)^{s-1}.$$

We need to show that the lemma holds for s examples. Consider the s-th example (x_s, ℓ_s). The probability that $\ell_1 + \cdots + \ell_s = (x_1 + \cdots + x_s) \cdot c$ is given by considering the cases where ℓ_s is either correct or incorrect. Specifically,

$$\begin{aligned}\Pr(\ell_1 + \cdots + \ell_s = (x_1 + \cdots + x_s) \cdot c) &= \Pr(\ell_s \text{ is correct}) \Pr(\ell_1 + \cdots + \ell_{s-1} = (x_1 + \cdots + x_{s-1}) \cdot c) \\ &+ \Pr(\ell_s \text{ is incorrect}) \Pr(\ell_1 + \cdots + \ell_{s-1} \neq (x_1 + \cdots + x_{s-1})\end{aligned}$$

By the induction hypothesis,

$$\begin{aligned}\Pr(\ell_1 + \cdots + \ell_s = (x_1 + \cdots + x_s) \cdot c) &= (1-\eta)\left(\frac{1}{2} + \frac{1}{2}(1-2\eta)^{s-1}\right) + \eta\left(\frac{1}{2} - \frac{1}{2}(1-2\eta)^{s-1}\right) \\ &= \frac{1}{2} + \frac{1}{2}(1-2\eta)^{s}.\end{aligned}$$

Thus, by induction, the lemma holds for all $s \geq 1$.

Example: Consider $s = 2$ and $\eta = 0.1$. The probability that $\ell_1 + \ell_2 = (x_1 + x_2) \cdot c$ is

$$\begin{aligned}\Pr(\ell_1 + \ell_2 = (x_1 + x_2) \cdot c) &= (1-0.1)\left(\frac{1}{2} + \frac{1}{2}(1-2\cdot 0.1)^{1}\right) + 0.1\left(\frac{1}{2} - \frac{1}{2}(1-2\cdot 0.1)^{1}\right) \\ &= 0.9\left(\frac{1}{2} + \frac{1}{2}(0.8)\right) + 0.1\left(\frac{1}{2} - \frac{1}{2}(0.8)\right) \\ &= 0.9\left(\frac{1}{2} + 0.4\right) + 0.1\left(\frac{1}{2} - 0.4\right) \\ &= 0.9 \cdot 0.9 + 0.1 \cdot 0.1 \\ &= 0.81 + 0.01 \\ &= 0.82 \\ \textit{equivalently}, & \\ &= \frac{1}{2} + \frac{1}{2}(1-2\cdot 0.1)^{2} \\ &= \frac{1}{2} + \frac{1}{2}(0.64) \\ &= 0.82.\end{aligned}$$

Therefore, the probability that $\ell_1 + \ell_2 = (x_1 + x_2) \cdot c$ is $\frac{1}{2} + \frac{1}{2}(1-2\eta)^2 = 0.82$.

Idea

The idea for the algorithm is to draw many more examples than the minimum needed to learn information-theoretically, we will be able to write basis vectors such as $(1, 0, \ldots, 0)$ as the sum of a relatively small number of training examples as $n^{O(\log\log n) >> 2^k}$.

- so length $O(\log n \log\log n)$ parity problem, we will be able to write $(1, 0, \ldots, 0)$ as the sum of only $O(\log n)$ examples.

- therefore, for any $\eta < \frac{1}{2}$, the corresponding sum of labels will be polynomially distinguishable from random.
- by repeating this process we may determine the correct label for $(1, 0, \ldots, 0)$, which is equivalently the first bit of the target vector c. This process can be further repeated to determine the remaining bits of c, allowing us to recover the entire target concept with high probability.

Algorithm for the Length-k Parity Problem

We view each example as consisting of a blocks, each b bits long (so, $k = ab$).

Definition 2

V_i is the subspace of $\{0,1\}^{ab}$ consisting of those vectors whose **last** i blocks have all bits equal to zero. An i-sample of size s is a set of s vectors independently and uniformly distributed over V_i.

The goal of our algorithm is to use labeled examples from $\{0,1\}^{ab}$ (these form a 0-sample) to create an i-sample such that each vector in the i-sample can be written as a sum of at most 2^i of the original examples. for all $i = 1, 2, \ldots, a-1$. We attain this goal via the following lemma.

Lemma 4

Assume we are given an i-sample of size s. We can in time $O(s)$ construct an $(i+1)$-sample of size at least $s - 2^b$ such that each vector in the $(i+1)$-sample is written as the sum of two vectors in the given i-sample.

Proof.

Let the i-sample be $x_1, \ldots, x_s$. In these vectors, blocks $a-i+1, \ldots, a$ are all zero. Each vector x_j can be written as:

$$x_j = [x_{j1}, x_{j2}, \ldots, x_{j(a-i)b}, \underbrace{0, 0, \ldots, 0}_{(i \text{ blocks}) \times b}]$$

$$\begin{array}{llllll}
 & \text{Block 1} & \text{...} \quad \text{Block j+1} & \cdots & \text{Block } a-i & \text{0 Blocks (i many)} \\
x_1 = & [x_{1,1}, x_{1,2}, & \ldots, x_{1,b}, \ldots, x_{1,jb+1}, x_{1,jb+1}, \ldots & & x_{1,(a-i-1)b+1}, \ldots, x_{1,(a-i)b} & , x_{1,(a-i)b+1}, \ldots, x_{1,ab}] \\
x_2 = & [x_{2,1}, x_{2,2}, & \ldots, x_{2,b}, \ldots, x_{2,jb+1}, x_{2,jb+1}, \ldots & & x_{2,(a-i-1)b+1}, \ldots, x_{2,(a-i)b} & , x_{2,(a-i)b+1}, \ldots, x_{2,ab}] \\
 & & & \ldots & & \\
x_s = & [x_{s,1}, x_{s,2}, & \ldots, x_{s,b}, \ldots, x_{s,jb+1}, x_{s,jb+1}, \ldots & & x_{s,(a-i-1)b+1}, \ldots, x_{s,(a-i)b} & , x_{s,(a-i)b+1}, \ldots, x_{s,ab}]
\end{array}$$

Partition $x_1, \ldots, x_s$ based on their values in block $a - i$. This results in a partition having at most 2^b classes. This is because each bit has 2 possible values and as there are b such bits in a block that means total permutations is 2^b.

From each **nonempty class** p, pick one vector x_{j_p} at random and add it to each of the other vectors in its class; then discard x_{j_p}. The result is a collection of vectors $u_1, \ldots, u_{s'}$, where $s' \geq s - 2^b$ (since we discard at most one vector per class).

considering that each $k \neq j$

$$\begin{tikzcd} x_{j_p} \arrow[d] \\ +x_{k_p} \arrow[d] \\ u_{k_p} = x_{j_p} + x_{k_p} \end{tikzcd}$$

So each partition now has 1 less vector as they for each partition x_{j_p} were discarded. the $u's$ form a subset in $V_i, u_1, \ldots, u_{s'}$ where each u_j is formed by summing two vectors in V_i which have identical components throughout block $a - i$, “zeroing out” that block. Each u_j is formed by taking some x_{j_p} and adding to it a random vector in V_i, subject to them being in same partition/ having vector agrees with x_{j_p} on block $a - i$. Therefore, each u_j is an **independent, uniform-random** member of V_{i+1}. The vectors $u_1, \ldots, u_{s'}$ thus form the desired $(i + 1)$-sample.

Proof of Theorem 2.

If we draw $a \cdot 2^b$ labeled examples. This means that we are expecting to cover all possible example of size ab, these qualify as a 0-sample. When Lemma 4 is applied $a-1$ times, to construct an $(a-1)$-sample it will have size at least 2^b and the vectors in it are distributed independently and uniformly at random over V_{a-1}, and notice that V_{a-1} contains only 2^b distinct vectors, one of which is $(1, 0, \dots, 0)$. Hence there is an approximately $1 - 1/e$ chance that $(1, 0, \dots, 0)$ appears in our $(a-1)$-sample. If this does not occur, we repeat the above process with new labeled examples. Note that the expected number of repetitions is only constant.

By unrolling the applications of Lemma 4, we can express the vector $(1, 0, \dots, 0)$ as the sum of 2^{a-1} labeled examples. By Lemma 3, we know that the sum of the labels of these examples gives us the correct value of $(1, 0, \dots, 0) \cdot c$ with probability

$$\frac{1}{2} + \frac{1}{2}(1 - 2\eta)^{2^{a-1}}.$$

To make the probability of error exponentially small, we repeat the entire process a polynomial number of times. Each repetition involves drawing new labeled examples and reapplying the algorithm.

By cyclically shifting all examples and repeating the algorithm, we can determine each bit of the target vector c using a number of examples and total computation time that is polynomial in $(\frac{1}{1-2\eta})^{2^a}$ and 2^b.

Finally, we choose $a = \frac{1}{2} \lg k$ and $b = \frac{2k}{\lg k}$. With these values, the bound on the number of samples and computation time becomes $2^{O(k/\log k)}$ for a constant noise rate η.

What kinds of functions can be learned from noisy, imperfect data efficiently?

3.1.3 k-wise version of SQ Model

In this set up instead of only being able to ask about statistical properties of a single example. (ex, Probability that a random example is labeled 1 given that its i-th bit is 1?) whereas the in k-wise the questions can be about properties of k-tuples of examples (what is the probability that two random examples have an even dot-product and have the same label?)

Remarks

- The Algorithm is *poly*(n) that is the total number of $(a_i, \tilde{y})$ pairs considered and not polynomial in the length of the secret vector.
- The algorithm does not require A to be random as long as the noise is random and there is no codeword within distance $o(n)$.
- Theorem 2 demonstrates that we can solve the length-n parity learning problem in time $2^{o(n)}$. However, this is achieved using $2^{O(n/\log n)}$ labeled examples, it would be beneficial to develop an algorithm that uses a number of examples that is polynomial in n Currently, it is unknown if this is possible.
- Consider the problem of learning parity functions that depend on the first k bits of input, where $k = O(\log n \log\log n)$:
  - If we set $a = \lceil \frac{1}{2} \log\log n \rceil$ and $b = O(\log n)$, the running time is polynomial in n.
  - The dependence on η is $\left(\frac{1}{1-2\eta}\right)^{\sqrt{\log n}}$, allowing us to handle η as large as $1/2 - 2^{-\sqrt{\log n}}$ and still have polynomial running time.

3.2 Implementation and Results of BKW Algorithm

3.2.1 Tools and Libraries Used

- **Python**: The programming language used for the implementation and a python notebook was used for code developement and experimentation.
- **NumPy**: For matrix and vector operations.

3.2.2 Implementation Details

*1. LPN Problem Generation The LPN problem instance is generated as follows:

- Randomly generate a secret vector $\mathbf{s} \in \{0,1\}^n$.
- Construct a random binary matrix $\mathbf{A} \in \{0,1\}^{m \times n}$.
- Compute the noiseless vector $\mathbf{b} = (\mathbf{A} \cdot \mathbf{s}) \mod 2$.
- Add noise to $\mathbf{b}$ with a given error rate.

Pseudocode:

```
GenerateLPNProblem(n, m, error_rate):
    s = RandomBinaryVector(n)
    A = RandomBinaryMatrix(m, n)
    noise = RandomBinaryVector(m, with_probability=error_rate)
    b = (A @ s) % 2
    b = (b + noise) % 2
    return A, b, s
```

2. BKW Algorithm

The BKW algorithm reduces the problem dimension by zeroing out bits iteratively using buckets. The steps are:

- Shuffle the equations.
- Perform bucket reduction to reduce dimensions using XOR operations.
- Apply Gaussian elimination to solve the reduced system.

Pseudocode:

```
BKWAlgorithm(A, b, bucket_size):
    Initialize buckets
    Shuffle rows of A and b
    for each row in A:
        Try reducing using existing buckets
        If reduced, add to corresponding bucket
        Otherwise, add to the first bucket
    Perform Gaussian elimination on reduced system
    Recover secret vector by back-substitution
    return recovered_s
```

3. Majority Voting

To improve robustness, the BKW algorithm is run multiple times, and the recovered secrets are aggregated using a majority voting system.

Pseudocode:

```
MajorityVoteBKW(A, b, bucket_size, runs):
    votes = []
    for i in range(runs):
        recovered_s = BKWAlgorithm(A, b, bucket_size)
        votes.append(recovered_s)
    final_s = MajorityVote(votes)
    return final_s
```

Results and Observations

The following results were obtained for $n = 8$, $m = 1024$, and an error rate of 0.2.

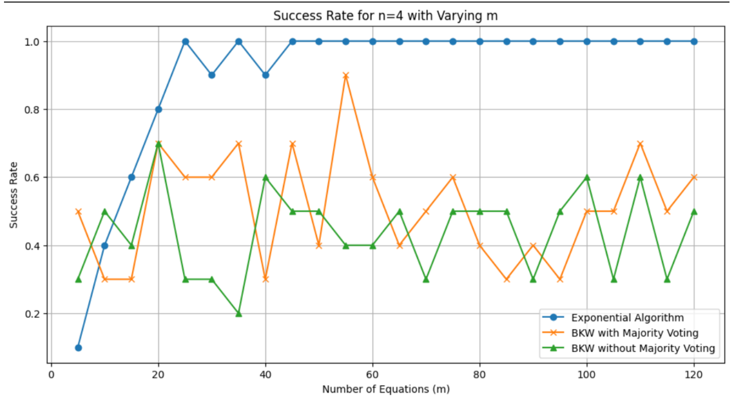
- **Original secret vector:** [0, 1, 0, 1, 1, 0, 1, 0]
- **Recovered secret vector (single run):** [0, 1, 0, 1, 1, 0, 1, 0]
- **Recovered secret vector (majority vote, 20 runs):** [0, 1, 0, 1, 1, 0, 1, 0]
- **Success Rate:** 100%

Conclusion

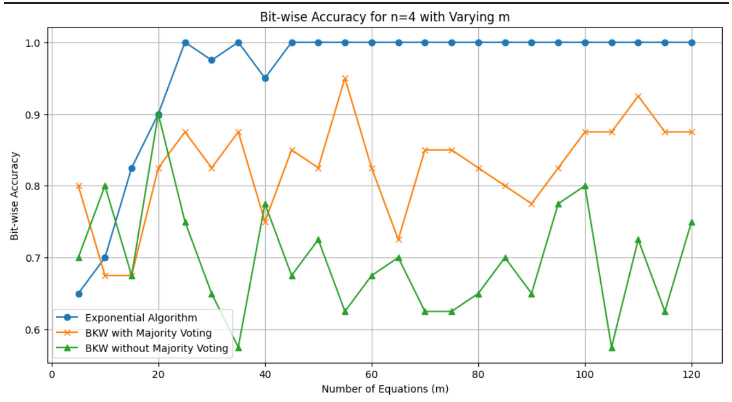
The BKW algorithm effectively recovered the secret vector from noisy equations. The majority voting mechanism further improved reliability. This demonstrates the practical applicability of the BKW algorithm for solving the LPN problem.

Comparison to Exponential Algorithm

Despite the low rate of success when the dimensionality of the problem gets higher and the number of samples are lower in relation, the BKW Algorithm gives results very similar to an exponential search for the key because of the use of Gaussian elimination instead of matrix multiplication over all possible candidates for the secret key. For lower dimension n=4 and ranging m with an error rate of 25% below were the results found for both the algorithms.



Success rates for BKW, BKW with majority and exp algorithm



Bit wise Accuracy for BKW, BKW with majority and exp algorithm

3.2.3 Future Work

Further work is required firstly in implementing the rolling system highlighted in the paper where instead of just zeroing out to create the basis/identity matrix one need to roll the vectors so as to always form a basis of the form [1,0,0,...,0]. This is still a work in progress and as progress would be made there would be updates in bothe the code files submitted and report. Work would also be done on the online algorithm described below.

Online Learning Algorithm Described in the paper

Multiple Matrices for Accuracy: - The algorithm uses multiple matrices (e.g., $M_1, M_2, \ldots$) to improve prediction accuracy. If an example passes through all matrices without being fully zeroed out, a majority vote from all matrices is used to predict the label. For a specific case with n examples, each k bits long (where $k = \frac{1}{4} \log n(\log\log n - 2)$ and $\eta = \frac{1}{4}$), the algorithm works as follows:

1. Example as Blocks: Each example consists of $(\log\log n - 2)$ blocks, each block having width $\frac{1}{4}\log n$.

2. Matrix Sequence: create a series of matrices $M_1, M_2, \ldots$.Initially, the matrices are empty.For each new example, if its first block doesn't match any row in M_1, it is added as a new row in M_1 and the algorithm says "I don't know." If it matches, the algorithm subtracts the row, zeroing out the first block, and considers the next block.

3. Handling New Rows: This process continues for all blocks of the example.If an example is written as the sum of rows in M_1, it is then added to the next matrix M_2, and so on, up to $M_{n^{2/3}}$.

4. Majority Vote:If an example passes through all matrices, a majority vote is used to predict the label with high probability. Each matrix can have at most $2^{\frac{1}{4}\log n(\log\log n - 2)}$ rows. The total number of examples where the algorithm fails to make a prediction is at most $n^{11/12} \log\log n = o(n)$.

Chapter 4

Arora-Ge for Learning Parities with Structured Noise

4.1 Problem Description

The *Learning Parities with Structured Noise (LPSN)* problem can be described as follows:

An oracle is associated with a polynomial P in m variables over $\mathbb{GF}(2)$. A noise vector $\eta \in \mathbb{GF}(2)^m$ is considered an *acceptable noise pattern* if it satisfies $P(\eta) = 0$.

Each time the oracle is queried:

1. The oracle randomly selects m vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m \in \mathbb{GF}(2)^n$.

2. The oracle then chooses an arbitrary acceptable noise pattern $\eta = (\eta_1, \eta_2, \ldots, \eta_m) \in \mathbb{GF}(2)^m$, such that $P(\eta) = 0$.

3. The oracle outputs:

$$(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m) \quad \text{together with the vector} \quad \mathbf{b} = (b_1, b_2, \ldots, b_m),$$

   where $b_i = \mathbf{a}_i \cdot \mathbf{u} + \eta_i$, for $i = 1, \ldots, m$, and $\mathbf{u} \in \mathbb{GF}(2)^n$ is a fixed but unknown secret vector.

- Let $P(\eta) = 0$ if and only if η is a vector of Hamming weight at most $m/3$. In this case, the polynomial corresponds to the intuitive notion of a noise rate of $1/3$. We explicitly construct and use such a polynomial in our implementation.

- More complex polynomials P can define other structured noise patterns. For example, P might encode constraints beyond simple Hamming weight restrictions. The problem of specifying a given distribution as a polynomial might be of independent interest.

Requirements for the Polynomial P: The polynomial P must satisfy the following:

1. There exists at least one $\rho \in \mathbb{GF}(2)^m$ such that $\rho \neq \eta + \eta'$ for all η, η' satisfying $P(\eta) = P(\eta') = 0$.
2. This condition ensures that the noise patterns introduced by the oracle do not completely overwhelm the signal encoded in $\mathbf{b}$.

The structured noise model allows the oracle to adaptively choose the noise pattern η, potentially considering the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$ in its decision. This is analogous to the agnostic-learning or worst-case noise model in the standard Learning Parity with Noise (LPN) problem, as discussed by Khot et al. Moreover, it generalizes the white noise case by introducing a richer set of constraints on the noise patterns via the polynomial P. We will, however, be working in the white noise case.

4.2 Arora and Ge's Algorithm for LPSN

Theorem 2.1: Suppose $P(\cdot)$ is such that there exists at least one $\rho \in \mathbb{GF}(2)^m$ such that $\rho \neq \eta + \eta'$ for all η, η' satisfying $P(\eta) = P(\eta') = 0$. Then there exists an algorithm that learns the secret $\mathbf{u}$ in poly(n, m) time.

Algorithm Steps: The algorithm works as follows:

1. **Input:**
   - Oracle $Q(\mathbf{u}, m, P, \mu)$, where:
     - $\mathbf{u} \in \mathbb{GF}(2)^n$ is the unknown secret vector,
     - m is the number of queries,
     - $P(\eta)$ is a degree-d polynomial over $\mathbb{GF}(2)$ defining the structured noise pattern,
     - μ is a distribution over noise vectors η satisfying $P(\eta) = 0$ (unknown to the algorithm).

- Access to poly$(N, 2^{m+d})$ samples from the oracle.

2. **Oracle Query:** Query the oracle Q to obtain:

$$(\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m), \quad (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_m),$$

where:

$$b_i = \mathbf{a}_i \cdot \mathbf{u} + \eta_i, \quad \eta = (\eta_1, \eta_2, \dots, \eta_m) \quad \text{and} \quad P(\eta) = 0.$$

3. **Construct Polynomial Constraints:** Using the oracle output, form the polynomial:

$$P(\mathbf{a}_1 \cdot \mathbf{z} + b_1, \mathbf{a}_2 \cdot \mathbf{z} + b_2, \dots, \mathbf{a}_m \cdot \mathbf{z} + b_m) = 0,$$

where $\mathbf{z}$ is an n-dimensional vector of variables. This results in a degree-d polynomial constraint in $\mathbf{z}$ that is always satisfied by the oracle's samples when $\mathbf{z} = \mathbf{u}$.

4. **Linearize the Polynomial:** Linearize the polynomial by introducing a new variable y_S for each monomial $\prod_{i \in S} z_i$, where $S \subseteq [n]$ and $|S| \leq d$. Replace:

$$\prod_{i \in S} z_i \quad \text{with} \quad y_S,$$

resulting in the linearized polynomial:

$$L\left(P(\mathbf{a}_1 \cdot \mathbf{z} + b_1, \mathbf{a}_2 \cdot \mathbf{z} + b_2, \dots, \mathbf{a}_m \cdot \mathbf{z} + b_m)\right) = 0.$$

5. **Form Linear System:** Collect poly$(N, 2^{m+d})$ samples to generate a system of linear equations:

$$\mathbf{A}\mathbf{y} = \mathbf{0},$$

where $\mathbf{A}$ is the matrix of coefficients obtained from the linearized constraints and $\mathbf{y}$ is the vector of variables y_S.

6. **Solve the Linear System:** Solve the linear system for $\mathbf{y}$. With high probability, the solution will satisfy:

$$y_{\{i\}} = u_i, \quad \text{for all } i \in [n].$$

7. **Output:** Extract the secret vector $\mathbf{u}$ from the solution $\mathbf{y}$.

Key Theoretical Results:

- The linear system has at least one solution, namely the tensor of the hidden vector $\mathbf{u}$:
$$y_S = \prod_{i \in S} u_i.$$

- With high probability, all solutions to the system will reveal $\mathbf{u}$, ensuring uniqueness of the recovered secret vector.

Theorem 2.2: The linear system formed by $10N2^{m+d}$ samples from the oracle always has at least one solution. With high probability, all solutions satisfy $y_{\{i\}} = u_i$ for all $i \in [n]$.

4.3 Implementation

Our first implementation in Python of the algorithm for solving the Learning Parities with Structured Noise (LPSN) problem had a few notable features.

- The implementation heavily relies on **SymPy**, a Python library for symbolic mathematics, to handle polynomial constraints and symbolic linearization.

- We construct a suitable polynomial explicitly. Consider m variables $x_1, x_2, \ldots, x_m$, where $x_i \in \mathbb{F}_2$. The *Hamming weight* of a vector $x = (x_1, x_2, \ldots, x_m)$ is defined as
$$\mathrm{wt}(x) = \sum_{i=1}^{m} x_i.$$
We construct a polynomial $P(x_1, x_2, \ldots, x_m) \in \mathbb{F}_2[x_1, x_2, \ldots, x_m]$ such that
$$P(x) = 0 \iff \mathrm{wt}(x) \leq \left\lfloor \frac{m}{3} \right\rfloor.$$
Define $S_k(x)$ to be the elementary symmetric polynomial of degree k, given by
$$S_k(x) = \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq m} x_{i_1} x_{i_2} \cdots x_{i_k}.$$

The desired polynomial $P(x)$ is then:

$$P(x) = \sum_{k=\lfloor m/3 \rfloor + 1}^{m} S_k(x).$$

This polynomial satisfies $P(x) = 0$ for all x with $\text{wt}(x) \leq \lfloor m/3 \rfloor$, and $P(x) \neq 0$ otherwise.

- The oracle function simulates the structured noise setting of the LPSN problem by:
  - Generating m random vectors $\mathbf{a}_i \in \mathbb{GF}(2)^n$.
  - Introducing structured noise into the responses $\mathbf{b}_i$ by selecting indices to corrupt based on the structured noise threshold.
- This design captures the essence of structured noise while remaining adaptable for different noise patterns.
- The function `key_constraint()` substitutes oracle outputs into the polynomial $P(\eta)$ to produce constraints involving the unknown $\mathbf{z}$.
- Polynomial terms are expanded and simplified modulo 2 to maintain consistency with operations in $\mathbb{GF}(2)$.
- The `linearize()` function converts degree-d polynomial constraints into linear constraints by introducing new variables for each monomial.
- The `gaussian_elimination()` function implements a symbolic solution for the linear system over $\mathbb{GF}(2)$.
- Constraints are represented as a matrix, and solutions are computed using `sp.linsolve()`, ensuring correctness in finite field arithmetic.

Due to the limitations of SymPy's solver, non-trivial solutions to the overdetermined system required algorithm could not be found. We noticed two important things. The first was the use of Gröbner bases by the SymPy solver, and the second was the suggestion in [1] of using Gröbner bases in the Arora-Ge algorithm. Our second implementation leverages SageMath, a robust mathematical software system, to solve the Learning Parities with

Structured Noise (LPSN) problem. The key difference from prior implementations lies in its reliance on **Gröbner bases** to solve the system of equations arising from the structured noise model. Below, we highlight the salient features of this approach:

Polynomial Ring and Gröbner Basis Computation

- A polynomial ring $\mathbb{F}_2[x_0, \ldots, x_{n-1}]$ is constructed with lexicographic ordering using SageMath's `PolynomialRing()` function.
- The equations, derived as polynomial constraints from the oracle outputs, are represented as elements of this ring.
- The Gröbner basis for the ideal generated by these polynomials is computed using `.groebner_basis()`, enabling systematic elimination of variables to reveal solutions.
- The lexicographic ordering ensures that the Gröbner basis computation reduces the equations to a triangular form, from which the solution can be directly read.

Gröbner Basis Solver

- The function `solve_groebner()` constructs the polynomial ring, converts the constraints into ring elements, and computes their Gröbner basis.
- The solution is extracted from the Gröbner basis as univariate linear polynomials of the form $x_i = c$, where $c \in \mathbb{F}_2$.
- This approach guarantees correctness for well-posed systems and is particularly suited to polynomial constraints over finite fields.

We describe the use of this tool from algebraic geometry in detail below, as we will reuse it in our implementation of the algorithm for LWE.

Gröbner Bases: Definition and Applications

Gröbner bases are a fundamental tool in computational algebra, particularly useful for solving systems of polynomial equations. Introduced by Buchberger in 1965, they generalize the notion of Gaussian elimination (for linear systems) and the Euclidean algorithm (for univariate polynomials) to multivariate polynomial systems. We refer the reader to [4] for a full treatment.

Definition of Gröbner Bases

Let $\mathbb{F}$ be a field, and let $R = \mathbb{F}[x_1, x_2, \ldots, x_n]$ be the polynomial ring in n variables over $\mathbb{F}$. Given an ideal $I \subseteq R$, a Gröbner basis for I is a finite subset $G = \{g_1, g_2, \ldots, g_t\} \subseteq I$ such that the leading term of every polynomial in I is divisible by the leading term of some $g_i \in G$.

The leading term of a polynomial depends on the choice of a monomial order (e.g., lexicographic order, graded reverse lexicographic order). A Gröbner basis is specific to the chosen monomial ordering.

Buchberger's Criterion: A Gröbner basis G satisfies the following condition:

$$\text{For all pairs } g_i, g_j \in G, \text{ the remainder of the } S\text{-polynomial of } g_i \text{ and } g_j \text{ upon division by } G \text{ is zero}$$

The S-polynomial of f and g is defined as:

$$S(f, g) = \frac{\mathrm{lcm}(\mathrm{lt}(f), \mathrm{lt}(g))}{\mathrm{lt}(f)} f - \frac{\mathrm{lcm}(\mathrm{lt}(f), \mathrm{lt}(g))}{\mathrm{lt}(g)} g,$$

where $\mathrm{lt}(f)$ denotes the leading term of f and lcm is the least common multiple.

Useful Properties of Gröbner Bases

1. Canonical Representation: Gröbner bases provide a canonical representation for the ideal I. This means two ideals are equal if and only if their Gröbner bases (with respect to the same monomial order) are equal.
2. Solution Extraction: If the Gröbner basis of I is triangular (i.e., each polynomial depends on fewer variables than the previous one in lexicographic order), solutions to the polynomial system $I = 0$ can be directly extracted.
3. Reduction: Any polynomial $f \in R$ can be reduced modulo the Gröbner basis G to a unique remainder r, where $r = 0$ if and only if $f \in I$.

Example: Solving a Polynomial System

Consider the system of equations:

$$f_1 = x^2 + xy - 2y^2 - 1 = 0,$$
$$f_2 = x^2 - y^2 - 2x = 0.$$

Let $R = \mathbb{Q}[x, y]$ with lexicographic ordering $x > y$. The ideal $I = \langle f_1, f_2 \rangle$ has a Gröbner basis G consisting of:

$$G = \{g_1 = y^2 + y - 1,\ g_2 = x + y^2 + 1\}.$$

The basis G is triangular. Solving $g_1 = 0$ gives the possible values of y, and substituting them into $g_2 = 0$ yields the corresponding values of x.

Application in the LPSN Problem

In the context of the LPSN problem, Gröbner bases are used as follows: 1. Polynomial System Construction: Constraints derived from the oracle are encoded as a system of multivariate polynomials over $\mathbb{F}_2$. 2. Gröbner Basis Computation: The ideal I generated by these polynomials is constructed, and its Gröbner basis G is computed using lexicographic order. 3. Solution Extraction: The triangular form of G allows direct recovery of the secret key $\mathbf{u}$ as the unique solution.

Advantages: - Gröbner bases provide a systematic and algebraically rigorous approach to solving polynomial systems. - They eliminate the need for explicit linearization by directly working with the full algebraic structure of the problem.

Computational Considerations

The computation of Gröbner bases is still, in general, computationally intensive, with complexity depending on the number of variables, the degree of the polynomials, and the chosen monomial ordering. In practice, tools like SageMath optimize these computations, leveraging sophisticated algorithms (e.g., F4/F5 algorithms) for efficiency.

4.3.1 Validation of Implementation

We are able to recover the secret key for n up to 10, beyond which the time taken by our implementation grows too excessive.

Chapter 5

Arora-Ge for Learning with Errors

The Learning with Errors (LWE) problem is a foundational problem in lattice-based cryptography. It involves recovering a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ given a sequence of approximate random linear equations that are corrupted by some small additive error. This problem is computationally hard and forms the basis of many cryptographic schemes.

5.1 Problem Description

Fix integers $n \geq 1$ (the dimension), $q \geq 2$ (the modulus), and let Ψ be a probability distribution on $\mathbb{Z}_q$ (the error distribution). The LWE problem is defined as follows:

- **Sample Generation:** For a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, draw a vector $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random and an error term $e \in \mathbb{Z}_q$ from the distribution Ψ.

- **Equation Construction:** Output the pair $(\mathbf{a}, b)$, where $b = \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q$. Here, $\langle \mathbf{a}, \mathbf{s} \rangle$ denotes the dot product of $\mathbf{a}$ and $\mathbf{s}$.

Given access to an arbitrary number of independent samples $(\mathbf{a}, b)$ generated as above, the goal is to recover the secret vector $\mathbf{s}$ with high probability.

Example: Suppose $n = 4$, $q = 17$, and Ψ generates errors in $\{-1, 0, 1\}$. Consider the following sequence of approximate linear equations:

$$\begin{aligned}
14s_1 + 15s_2 + 5s_3 + 2s_4 &\approx 8 \pmod{17},\\
13s_1 + 14s_2 + 14s_3 + 6s_4 &\approx 16 \pmod{17},\\
6s_1 + 10s_2 + 13s_3 + 1s_4 &\approx 3 \pmod{17},\\
10s_1 + 4s_2 + 12s_3 + 16s_4 &\approx 12 \pmod{17},\\
9s_1 + 5s_2 + 9s_3 + 6s_4 &\approx 9 \pmod{17},\\
3s_1 + 6s_2 + 4s_3 + 5s_4 &\approx 16 \pmod{17}.
\end{aligned}$$

Here, the "" denotes equality up to a small additive error (e.g., ± 1). The solution to this system is $\mathbf{s} = (0, 13, 9, 11)$.

Formal Definition

The LWE problem can be formally defined in terms of a distribution. Define the distribution $A_{\mathbf{s},\Psi}$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ as follows:

$A_{\mathbf{s},\Psi}$ samples $(\mathbf{a}, b)$ where $\mathbf{a} \in \mathbb{Z}_q^n$ is uniform, $b = \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod{q}$, and $e \sim \Psi$.

An algorithm solves the LWE problem with modulus q and error distribution Ψ if, for any $\mathbf{s} \in \mathbb{Z}_q^n$, it outputs $\mathbf{s}$ with high probability given an arbitrary number of independent samples from $A_{\mathbf{s},\Psi}$.

Connections and Special Cases

1. **Learning Parity with Noise (LPN):** When $q = 2$, the LWE problem reduces to the well-known *Learning Parity with Noise (LPN)* problem. Here, the error distribution Ψ is typically Bernoulli, with a parameter $\varepsilon > 0$ representing the probability of error. 2. **Decoding Random Linear Codes:** LWE can be viewed as a problem of decoding from random linear codes in the presence of noise. 3. **Lattices and Bounded Distance Decoding:** LWE is also equivalent to a bounded distance decoding problem on random integer lattices, further connecting it to hard problems in lattice theory.

Cryptographic Importance

The LWE problem's hardness is supported by reductions from worst-case lattice problems such as the Shortest Vector Problem (SVP) and the Shortest

Independent Vectors Problem (SIVP). This hardness assumption underpins the security of many cryptographic primitives, including public-key encryption, key exchange protocols, and fully homomorphic encryption schemes.

Below, we describe the sample generation process in detail.

Discrete Gaussian Noise Generation The noise term η in the LWE problem is generated according to the **Discrete Gaussian Distribution** Ψ_α with standard deviation αq, where α is a parameter controlling the noise level relative to the modulus q. The process is as follows:

- Pick a random real number r from the continuous Gaussian distribution with standard deviation $\sigma = \alpha q$, whose density function is:

$$D_\sigma(r) = \frac{1}{\sigma} \exp\left(-\pi \left(\frac{r}{\sigma}\right)^2\right).$$

- Round r to the nearest integer $\lfloor r \rceil$, where $\lfloor \cdot \rceil$ denotes rounding to the closest integer.
- Reduce the result modulo q to obtain $\eta \in \mathbb{Z}_q$, i.e., $\eta = \lfloor r \rceil \pmod{q}$.

This method ensures that the noise is distributed symmetrically and is concentrated around 0, with larger deviations occurring with exponentially smaller probability. The parameter α governs the trade-off between noise magnitude and problem hardness.

Oracle Access for Sample Generation The LWE problem assumes access to an oracle $Q(\mathbf{u}, \Psi_\alpha)$, which produces noisy linear equations based on a secret vector $\mathbf{u} \in \mathbb{Z}_q^n$. The oracle works as follows:

- Choose $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random.
- Independently sample noise $\eta \in \mathbb{Z}_q$ from the distribution Ψ_α.
- Output the pair $(\mathbf{a}, b)$, where:

$$b = \langle \mathbf{a}, \mathbf{u} \rangle + \eta \pmod{q}.$$

Conjectured Hardness It is conjectured that no polynomial-time algorithm can efficiently recover the secret $\mathbf{u}$ from the samples provided by $Q(\mathbf{u}, \Psi_\alpha)$ for properly chosen parameters n, q, and α. The best-known algorithms for solving LWE, under such parameter choices, run in exponential time, highlighting the problem's computational difficulty.

This careful choice of noise distribution is critical in ensuring the LWE problem's cryptographic hardness, as the discrete Gaussian noise introduces just enough uncertainty to render linear-algebraic techniques ineffective while maintaining enough structure for theoretical analysis.

Algorithm for Solving LWE

We describe the subexponential algorithm due to Arora and Ge for solving the Learning with Errors (LWE) problem under a model similar to the structured noise setting in Chapter 2. This algorithm relies on the use of a modified oracle $Q(u, \Psi_\alpha, d)$, which ensures that the noise η is bounded by a parameter d. The algorithm combines ideas from Gaussian noise analysis and linearization techniques, extending methods used for the Learning Parity with Noise (LPN) problem.

Theorem 3.1 When $\alpha q = n^\varepsilon$ (where ε is a constant strictly smaller than $\frac{1}{2}$) and $q \gg (\alpha q \log n)^2$, there exists a $2^{\tilde{O}(n^{2\varepsilon})}$-time algorithm that learns u given access to the oracle $Q(u, \Psi_\alpha)$.

The structured noise model ensures that η is sampled uniformly from Ψ_α conditioned on $|\eta| \leq d$, where $2d + 1 < q$.

Preliminaries We rely on the following properties of the Gaussian distribution Ψ_α (Lemma 3.2):

1. For $\eta \in \left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$,
$$\Pr_{\eta \sim \Psi_\alpha} [|\eta| > k\alpha q] \leq e^{-O(k^2)}.$$

2. For $\eta \in \Psi_\alpha$,
$$\Pr_{\eta \sim \Psi_\alpha} [\eta = 0] = \Omega\left(\frac{1}{\alpha q}\right).$$

Algorithm Description [1] Oracle $Q(u, \Psi_\alpha, d)$, n-dimensional secret u, modulus q, noise parameter αq, and bound d satisfying $2d + 1 < q$. Secret vector u. Define a univariate polynomial $P(\eta)$ of degree $D = 2d + 1$ such that $P(\eta) = 0$ whenever η is sampled from $Q(u, \Psi_\alpha, d)$:

$$P(\eta) = \eta \prod_{i=1}^{d} (\eta + i)(\eta - i).$$

Substitute $\eta = \mathbf{a} \cdot \mathbf{z} + b$ into $P(\eta)$ to obtain a polynomial constraint:

$$(\mathbf{a} \cdot \mathbf{z} + b) \prod_{i=1}^{d} (\mathbf{a} \cdot \mathbf{z} + b + i)(\mathbf{a} \cdot \mathbf{z} + b - i) = 0,$$

where $\mathbf{z}$ is an n-dimensional variable vector. Linearize the polynomial $P(\mathbf{a} \cdot \mathbf{z} + b)$ using a variable vector $\mathbf{y}$ indexed by vectors $\mathbf{v} \in \mathbb{Z}^n$ such that $1 \leq \sum_{i=1}^{n} v_i \leq D$. The variable $y_{\mathbf{v}}$ corresponds to the monomial $\prod_{i=1}^{n} z_i^{v_i}$. Construct a linearization operator $\mathcal{L}$ that replaces each monomial in $P(\mathbf{a} \cdot \mathbf{z} + b)$ with the corresponding y-variable. The resulting equation is:

$$\mathcal{L}\left(P(\mathbf{a} \cdot \mathbf{z} + b)\right) = 0.$$

Query the oracle $Q(u, \Psi_\alpha, d)$ $O(N \alpha q^2 \log q)$ times, where $N = \binom{n+D}{n}$ is the number of y-variables. Solve the resulting system of linear equations over the y-variables. Extract u from the solution by identifying $y_{\mathbf{e}_i} = u_i$, where $\mathbf{e}_i$ is the vector with 1 in the i-th coordinate and 0 elsewhere.

Theorem 3.3 With high probability, all solutions to the system of linear equations generated as above satisfy $y_{\mathbf{e}_i} = u_i$, thereby recovering u.

Complexity The algorithm runs in time $2^{\tilde{O}(n^{2\varepsilon})}$, which is subexponential for small ε.

5.2 Implementation Details

The provided code implementation employs various algorithmic and computational techniques to solve the Learning with Errors (LWE) problem. Below, we outline the key features and their significance.

Modular Arithmetic Handling: We include a custom `mod` function to match the C99 implementation, which makes `mod` consistent with rounding towards 0 which is preferred since it is rather natural to expect the result of -a / b be the same as -(a / b). In case of true modulo behavior, -1 % 10 would evaluate to 9, meaning that -1 / 10 has to be -1. This in particular helps us generate errors centered at 0.

Polynomial Representation of Constraints: The algorithm uses SageMath's `PolynomialRing` to construct polynomial representations of the LWE problem:

- Each generator variable x_i represents a component of the secret vector $\mathbf{s}$.
- For each LWE sample, a polynomial is constructed as:

  $$P_i = \prod_{e \in E} \left(b[i] - \sum_j A[i][j] x_j - e \right),$$

  where E is the set of possible noise values. This polynomial models the noisy linear equation constraint.

Groebner Basis for Solution Recovery: The polynomials are added to an ideal in the polynomial ring, and the algorithm computes the Groebner basis of this ideal. The Groebner basis serves to simplify the system of polynomials, isolating linear terms that can be directly solved for the secret:

- Each resulting polynomial in the basis corresponds to a single variable x_i.
- The constant term of the linear polynomial yields the recovered value of the corresponding secret component.

Efficient Error Modeling: The set E models structured noise using a range of integers, both positive and negative. The algorithm's design accommodates non-trivial distributions for the noise, making it flexible for various LWE instances. Gaussian sampling is usually the most expensive step for algorithmic attacks against LWE.

Prime Modulus Selection: The modulus q is chosen as the nearest prime to:

$$q = \lceil 100 \cdot (\sigma \cdot \log_2 n)^2 \rceil,$$

ensuring q is large enough to accommodate the noise and secret vector dimensions while being computationally efficient for modular arithmetic.

5.3 Simplified Proof Explanation for Theorem 3

The essence of the proof relies on representing an LWE sample $(\mathbf{a}, b := \mathbf{a}^T\mathbf{s} + e)$, where $e \in S \subseteq \mathbb{Z}_q$, as a polynomial equation:

$$f_{\mathbf{a},b}(\mathbf{s}) = \prod_{x \in S}(b - \mathbf{a}^T\mathbf{s} - x) \mod q,$$

where b and $\mathbf{a}$ are known, while $\mathbf{s}$ is the unknown secret. By construction, $f_{\mathbf{a},b}(\mathbf{s}) = 0 \mod q$ if and only if $(\mathbf{a}, b)$ is an LWE sample. Solving the system of polynomial equations

$$f_{\mathbf{a}_i,b_i}(\mathbf{s}) = 0 \mod q \quad \text{for } i = 1, \ldots, m,$$

yields the secret vector $\mathbf{s}$. However, solving systems of polynomial equations, even those of degree 2, is known to be NP-hard.

Arora and Ge's key insight was that if there are sufficiently many equations, the system can be linearized such that the solution to the resulting linear system also solves the polynomial system with high probability (w.h.p.).

To understand this, consider the degree of the polynomials $f_{\mathbf{a},b}(\mathbf{s})$, which is $|S|$ (the size of the domain for the noise terms). The number of monomials in $f_{\mathbf{a},b}(\mathbf{s})$ is $\binom{n+|S|}{|S|}$. Linearization replaces each monomial with a new variable. For $m \gg \binom{n+|S|}{|S|}$, there are more equations than variables, and any solution to the polynomial system is also a solution to the linearized system. When m is large enough, the solution $\mathbf{s}$ becomes unique w.h.p.

Illustrative Case: To simplify, assume $S = \{0, 1\}$ and $n = 1$. Each LWE sample $(a, b = a \cdot s + e)$, where $e \in \{0, 1\}$ and $a, s \in \mathbb{Z}_q$, gives the polynomial:

$$(b + a \cdot s) \cdot (b + a \cdot s - 1) = 0 \mod q.$$

Expanding this yields:

$$b(b-1) + (2b-1)a \cdot s + a^2 \cdot s^2 = 0 \mod q.$$

Linearizing involves introducing independent variables u_1 and u_2 to replace s and s^2, respectively:

$$p(a) = b(b-1) + (2b-1)a \cdot u_1 + a^2 \cdot u_2 = 0 \mod q.$$

Initially, one might argue that no (u_1, u_2) satisfies this equation w.h.p. over $a \leftarrow \mathbb{Z}_q$. However, this is incorrect, as $u_1 = s$ and $u_2 = s^2$ always form a valid solution.

Instead, substitute $b = a \cdot s + e$ into the linearized equation:

$$p'(a) = e(e-1) + (2e-1)(s+u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a^2 = 0 \mod q.$$

Since $e(e-1) = 0$ by definition, this simplifies to:

$$p'(a) = (2e-1)(s+u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a^2 = 0 \mod q.$$

Here, $p'(a)$ is viewed as a polynomial in a with coefficients independent of a. Fix any (u_1, u_2) such that $u_1 \neq -s$. In this case, $p'(a)$ is a non-zero polynomial in a, and the probability it equals zero is at most $2/q$ by an application of Cauchy-Schwarz. Using a Chernoff bound and union bound over all samples, we conclude that (u_1, u_2) does not satisfy the system w.h.p., leaving $(-s, s^2)$ as the unique solution.

This argument establishes the feasibility of recovering $\mathbf{s}$ by solving the linearized system, leveraging the statistical properties of the polynomial coefficients.

5.4 Validation of the Implementation

We are able to recover the secret key for n up to 3, beyond which the time taken by the algorithm grows too excessive. In [1], a time of 68 hours is reported for recovering a key of size 10.

Chapter 6

Conclusion, Discussion, and Future Directions

While we would have liked to have produced a nice looking curve for the time required by the algorithm and the changing parameter size, the explosion in time taken between a small and large value made this impossible to do for a meaningfully wide interval of parameter values.

It is natural to ask whether the Arora-Ge method can solve the Learning Parity with Noise (LPN) problem. We discern that the approach fails for LPN due to the inherent properties of the modulus $q = 2$, which prevent the method from uniquely identifying the secret s or even narrowing down the possible solutions.

The failure arises in the algorithm's first step, where each LPN sample is converted into a quadratic equation in s that encodes the condition $\langle \mathbf{a}_i, \mathbf{s} \rangle = b_i \mod 2$. These equations are then linearized and solved for $\mathbf{s}$. However, any candidate $\mathbf{s}'$ satisfies these equations, as the right-hand side is always 0 mod 2. Consequently, the system provides no meaningful constraints on $\mathbf{s}$, rendering the algorithm ineffective for LPN.

In contrast, the method is effective for Learning with Errors (LWE) when the modulus q is larger than the number of possible error values. Under this condition, the equations $\langle \mathbf{a}_i, \mathbf{s} \rangle = b_i \mod q$ impose nontrivial constraints on $\mathbf{s}$. With a sufficient number of such equations, the system can uniquely determine $\mathbf{s}$, enabling the Arora-Ge technique to succeed for LWE but not for LPN.

For future work, the prospect of coming up with any algorithmic improvements to the work we've studied is an obvious extension. Beyond that, we can ask a couple well-posed problems. In the case of LPN, Arora-Ge needs $m = W(n^2)$ LWE samples. It remains a tantalizing open problem to come up with a more sample-efficient attack or prove that doing so is hard. A concrete way to demonstrate the latter would be to show that solving binary error LWE with $o(m^2)$ samples is as hard as solving the lattice (approximate) shortest vector problem.

Another open question that captures our interest is to characterise the distributions of the secret s for which the LWE assumption holds (assuming LWE with uniform secrets holds).

GitHub link:https://github.com/niyati-rao/Capstone_Project_LWE

Appendix A

Definitions and simple proofs

PAC model

The **PAC (Probably Approximately Correct)** model is a framework for how well a learning algorithm can approximate a target function with high probability, given enough training data. An algorithm is said to learn in the PAC model if, with high probability $(1-\delta)$, the hypothesis it outputs has low error (ϵ) on unseen data, as long as it is provided with a sufficient number of examples.

The **Statistical Query (SQ)** model, a restricted version of the PAC model, only allows the algorithm to query statistical properties rather than directly observing individual labels. Problems that can be solved using the SQ model can be considered as part of the PAC model, since any learning algorithm that works in the SQ model also works in the PAC model, but the reverse is **not true**.

A.1 Statistical Query (SQ) Model

The Statistical Query (SQ) model, introduced by Michael Kearns in 1993, is a framework for studying machine learning algorithms that interact with data through statistical queries instead of direct data access. This model helps design robust algorithms that can tolerate noise and errors.

- **Idea:** Requests for approximate values of statistical properties of the data distribution. Therefore there is no direct data point but general-

ized structures assuming errors as we only have approximate answers to statistical queries.
class of parity functions without noise can efficiently be learned in the PAC model and provably cannot be learned in the PAC model.[cite]

- **Tolerance Parameter**: Specifies the accuracy/noise parameter level in the oracle's answers. $\tau \in [0, 1]$. The query returns an approximate probability $\hat{P}_Q \in [P_Q - \tau, P_Q + \tau]$, where $P_Q = \Pr_{x \leftarrow D}[Q(x, c(x))]$.

- **SQ Dimension**: Measures the complexity of a concept class in the SQ model.

- **Types of queries:** Expectation Query $\mathbb{E}_{\mathbf{x} \sim \mathcal{D}}[f(\mathbf{x})]$,Probability of event data distribution: $\Pr_{\mathbf{x} \sim \mathcal{D}}[A(\mathbf{x})]$, Variance and covariance Query, or queries regarding any property Q of labeled examples (e.g., "the first two bits are equal and the label is positive"),

  Constraint: Each query Q must be **polynomially evaluable**, meaning it can be computed in polynomial time given (x, ℓ).

- A statistical query can be simulated with a large sample of data by computing an empirical average, requiring about $O(1/\tau^2)$ samples for accuracy τ. [A.3]

- A concept class C is **learnable from statistical queries** under distribution D if there exists an algorithm A that can produce an ϵ-approximation of $c \in C$ in polynomial time, using polynomially many queries and tolerance. **Weak learning**is when an algorithm weakly learns C by achieving an approximation $\epsilon < \frac{1}{2} - \frac{1}{\text{poly}(n)}$, meaning it does better than random guessing.

- The SQ model assumes a **known distribution** D; in an **unknown distribution** setting, the algorithm can query for unlabeled examples from D to ensure robust learnability.

SQ learning applies to **non-noisy data**, but statistical queries can be simulated with noisy data, so any concept learnable in SQ is also PAC-learnable with random classification noise. Therefore, an algorithm for learning in the in the SQ Model can automatically be converted to an algorithm for learning in the presence of noise in PAC model.

A.2 noise rate = 1/4, then the sum of s labels has noise rate $1/2(1/2)^{s+1}$

Single Bit Flips:

For a single label, there is a $\frac{1}{4}$ chance it is incorrect. Let's denote this probability as

$$\eta = \frac{1}{4}.$$

Summing Two Labels:

When summing two binary variables (mod 2), if both are correct or both are incorrect, their sum (mod 2) will be correct. The probability that both labels are correct is

$$\left(\frac{3}{4}\right)^2 = \frac{9}{16}.$$

The probability that both labels are incorrect is

$$\left(\frac{1}{4}\right)^2 = \frac{1}{16}.$$

Therefore, the probability that the sum is correct is

$$\frac{9}{16} + \frac{1}{16} = \frac{10}{16} = \frac{5}{8}.$$

This means that the sum of two labels has a noise rate of

$$1 - \frac{5}{8} = \frac{3}{8}.$$

Summing More Labels:

We know that the sum will be correct as long as as an even number of examples are marked wrong. Thid can then be seen as the sum of geometric progression with finite terms. The formula for the sum of the first n terms is:

$$S_n = \frac{a(1-r^n)}{1-r} \text{if } r \neq 1, = \frac{1}{4}\frac{1-\left(\frac{1}{2}\right)^n}{1-\frac{1}{2}}$$

Simplifying the denominator:

$$S_n = \frac{1}{4}\frac{1-\left(\frac{1}{2}\right)^n}{\frac{1}{2}} = \frac{1}{2}\left(1-\left(\frac{1}{2}\right)^n\right) = \frac{1}{2} - \left(\frac{1}{2}\right)^{n+1}$$

The pattern observed leads to the general result:

When summing s labels, the combined noise rate approaches $\frac{1}{2}$, adjusted by a diminishing factor specifically, for a noise rate of $\frac{1}{4}$ in the original data, the combined noise rate for the sum of s labels becomes:

$$\text{Noise Rate} = \frac{1}{2} - \frac{1}{2^{s+1}}.$$

A.3 $O(1/\tau^2)$ samples for accuracy τ

the empirical average of property Q serves as an approximation of the true probability $P_Q = \Pr[Q(x, c(x))]$.

- Assume each sample observation of Q is a random variable with variance σ^2.
  - The standard error (standard deviation of the sample mean) decreases as $\frac{\sigma}{\sqrt{m}}$ where m is the number of samples.
  - To ensure it is at most τ, we set $\frac{\sigma}{\sqrt{m}} \leq \tau$, which implies $m \geq \frac{\sigma^2}{\tau^2}$.
- σ^2 of Q is bounded ($\sigma^2 \leq 1$ for boolean functions), the number of samples m required for the empirical average to approximate P_Q within an accuracy of τ is:

$$m = O\left(\frac{1}{\tau^2}\right).$$

Bibliography

[1] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for lwe problems. *ACM Commun. Comput. Algebra*, 49(2):62, August 2015.

[2] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 403–415, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.

[4] David Cox, John Little, and Donal O'Shea. Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra. 2007.