CS 354: Unfulfilled Algorithmic Fantasies May 15, 2019

Lecture 14: Learning with Errors

Lecturer: Aviad Rubinstein *Scribe: Nick Guo*

1 Learning with Errors

This lecture will introduce the Learning with Errors (LWE) problem.

1.1 LWE Definition

We receive input pairs (a_i, b_i) where

$$a_i \sim \mathbb{Z}_q^n \text{ is a vector sampled uniformly at random}$$
$$b_i \leftarrow a_i \cdot s + \text{noise} \pmod q \text{ is a scalar in } \mathbb{Z}_q$$

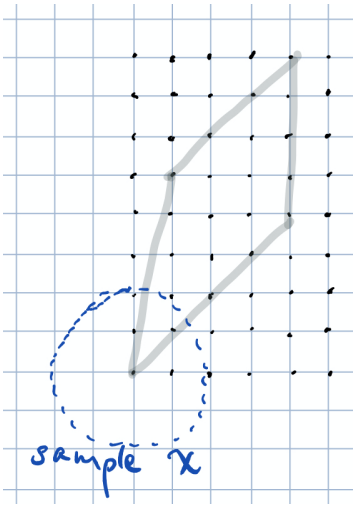
Our objective is to find the vector s that works for all input pairs. It's similar to learning linear equations, but we have a little bit of noise. If we didn't have the noise, then finding s would be very easy: after about n pairs, we can recover s in polynomial time using Gaussian elimination.

1.2 Connections to Lattices

We will connect LWE to lattice problems. Recall SVP (Shortest Vector Problem) seen in the prior lecture; in that setting, we were trying to find the vector in the lattice with smallest norm that wasn't $\vec{\mathbf{0}}$, and this is equivalent to finding the closest vector to $\vec{\mathbf{0}}$.

In this lecture, we will be working with CVP (Closest Vector Problem) which is similar to SVP. The input to CVP is a vector $x \notin \Lambda$, and the output is $x' \in \Lambda$ such that x' is the closest to x.

Lemma 1.1. *CVP(x)* $= \vec{\mathbf{0}} \iff$ *SVP of the lattice is large.*



We sample x, and then try to find CVP(x).
Intuitively, if the lattice doesn't have any short vectors, then the closest thing to x would just be $\vec{\mathbf{0}}$ if x is small. However, if the lattice has small shortest vectors, then the closest element from the lattice to a small x is probably not $\vec{\mathbf{0}}$.

1.3 Hardness of LWE

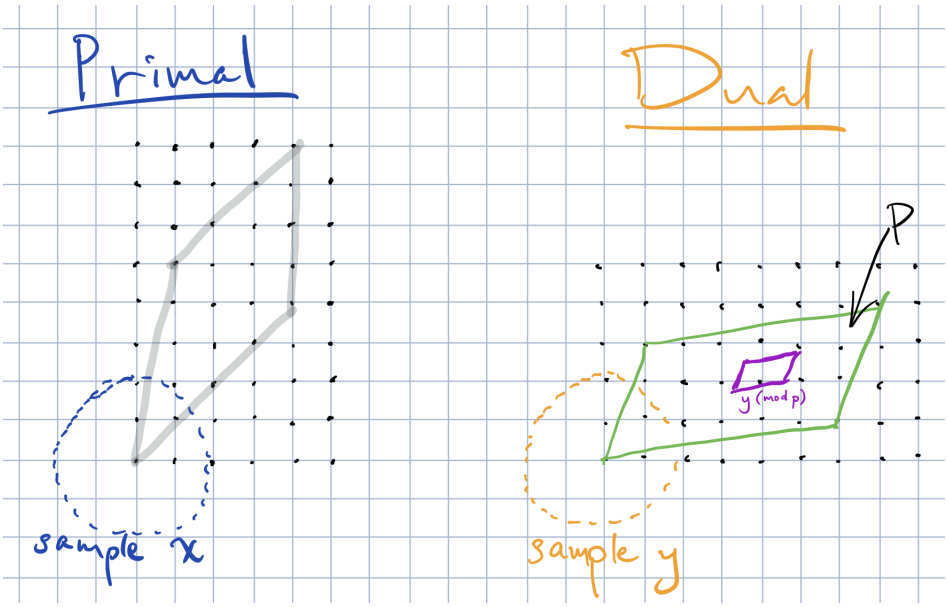
Theorem 1.2 ([Reg05, Pei09])**.** *If gap (approximate) SVP is hard in the worst-case, then LWE is hard in the average case.*

Sketch of Proof Sketch. Similar to proofs in lecture 13, we will be working the primal lattice Λ and the dual lattice Λ^*. Recall the definition of the dual lattice:

$$\Lambda^* := \{y \in \mathbb{R}^n \mid \forall x \in \Lambda, x^T y \in \mathbb{Z}\}$$

Let the primal lattice be the original lattice from CVP, and let x be a sample that is not in Λ. Let y be a sample from the dual lattice divided by q (tile the parallelepiped with q^n sub-parallelepipeds, q for each dimension of the lattice). We also require that y be within a ball around the origin, and that it be at the corner of one of these sub-parallelepipeds: $y \in \Lambda^*/q$.

Finding small vectors in the dual is hard (equivalent to finding small vectors in the primal), so we make this compromise on y since it's not exactly in the dual (it's in the dual divided by q). Rounding down to a sub-parallelepiped allows us to sample y from the ball since it is computationally hard to find a true sample from Λ^* inside the ball. Note that although y is small, when we take $y \pmod P$ where P is the original parallelepiped in Λ^*, it is a potentially large vector since the difference is large (because we don't have a very good basis, P contains large vectors).



Our goal is to reduce SVP to LWE. We will sample x once, and y multiple times (each pair we feed to the LWE instance will use a different y_i). Consider the following settings for a_i and b_i:

$$a_i = qB^{-1}(y_i \pmod p)$$
$$b_i = q(y_i \cdot x) + \text{noise} \pmod q$$

a_i represents the index of $y_i \pmod P$'s sub-parallelepiped; each dimension of a_i is how many small tiles we need to go in the corresponding direction from the origin to $y_i \pmod p$. The dual lattice's

basis is represented by B^{-1}.

We will now demonstrate what the s is for $b_i = a_i \cdot s$. We use B^* as the transpose of Λ's basis.

$$\begin{aligned} b_i = q(y_i \cdot x) + \text{noise} \pmod q &\approx q(y_i \cdot \text{CVP}(x)) \pmod q && (1)\\ &= q(y_i (\text{mod } P) \cdot \text{CVP}(x)) \pmod q && (2)\\ &= [qB^{-1} y_i (\text{mod } P)] \cdot [B^* \text{CVP}(x)] \pmod q && (3)\\ &\approx a_i \cdot s \pmod q && (4) \end{aligned}$$

- (1) follows from the fact that y_i is small, and x is close to $\text{CVP}(x)$ by definition of CVP.
- (2) used the fact that $y_i - y_i (\text{mod } P) \in \Lambda^*$.

$$\begin{aligned} y_i - y_i (\text{mod } P) \in \Lambda^* &\implies \big(y_i - y_i (\text{mod } P)\big) \cdot \text{CVP}(x) \in \mathbb{Z} && \text{(By def. of dual lattice)}\\ &\implies q\big(y_i - y_i (\text{mod } P)\big) \cdot \text{CVP}(x) \equiv 0 \mod q\\ &\implies q(y_i \cdot \text{CVP}(x)) \equiv q\big(y_i (\text{mod } P)\big) \cdot \text{CVP}(x) \mod q \end{aligned}$$

- (3) is simply applying and then undoing a vector multiplication with the inverse basis.
- (4) substitutes the definition of a_i in the left expression, and we see that $s = B^* \text{CVP}(x)$.

The main idea behind the proof is that if we have an efficient algorithm for approximate LWE, then we can learn the s value. Using s, we're able to compute $\text{CVP}(x)$ by multiplying with B^{-1}. Therefore, if $\text{CVP}(x)$ is hard, then LWE is also hard.

1.4 Contribution of Noise

We've left out some hard details in the proof sketch of theorem 1.2 because we did not account for the contribution of the noise. The amount of noise we can handle is directly related to how large q is. If the noise is larger than q, then it will wrap around and just not make sense since we are operating modulo q.

Another question is how small is y? In (1), we relied on y being small and the presence of the noise to allow for us to go from x to $\text{CVP}(x)$.
In [Pei09], it's shown that we can take q, $||y||$, $||\text{noise}||$ to be exponential in n, and the proof still works.

In [Reg05], we take q to be $\text{poly}(n)$. The size of the noise depends on the size of y, which is in the dual lattice divided by q. Maybe the dual lattice has small vectors, but we don't know how to find them. So at the very least, if we want to take something that we can actually sample, we just use the basis vectors divided by q (there may be a combination of basis vectors resulting in a smaller vector, but at the very least we can use the basis vectors directly). Therefore we see that:

$$||\text{noise}|| \approx ||y|| \approx \frac{||\text{basis vector}||}{q}$$

The basis vector is much worse (larger) than the actual best vector to sample using, and q is only a polynomial in n, so the y that we sample is too large and won't work for the proof.

With so much noise, we cannot find CVP(x) in one shot, but what we can find is $x' \in \Lambda$ (which may not be as good as CVP(x)) that is roughly a q factor closer to x than the trivial vector. When we do this, we learn something about the lattice (nontrivial vector) that we can then use to do a better job of sampling y so that it's a little smaller than the y that we sampled earlier (by roughly a q factor).

The details of how to do this are omitted, but now we can recursively apply this process and eventually gain an exponential factor in reducing the size of y. The catch is that to sample the smaller y, we need to use a quantum algorithm. In this part, we assume that we have an LWE algorithm, and call it a bunch of times (such an algorithm doesn't necessarily exist). Thus, the hardness assumption is that approximate SVP is hard even for quantum computers.

2 Applications of LWE

The most interesting applications of LWE are in cryptography.

2.1 Fully Homomorphic Encryption

Suppose you have a function (supports additions and multiplications) that you wish someone else to compute for you, but you don't want them to know the input to the function. You can encrypt the input, and the other party operates on the encryption. Finally, you take their output and decrypt it to get the result. Amazingly, it turns out that you can do it using LWE [Bra18].

Here's some intuition. Suppose you want to add $s_1 + s_2$. We can encrypt s_1 by doing $\text{Enc}(s_1) = (a_i, a_i \cdot s_1 + \text{noise})_{i=1}^n$, and s_2 by $\text{Enc}(s_2) = (a_i, a_i \cdot s_2 + \text{noise})_{i=1}^n$. Notice that this exactly resembles an LWE instance, and because LWE is assumed to be hard, you cannot learn s_1 and s_2.

However, we see that if we have the other add up the encryptions, the result is

$$\text{Enc}(s_1) + \text{Enc}(s_2) = (a_i, a_i \cdot s_1 + a_i \cdot s_2 + \text{ noise})_{i=1}^n = (a_i, a_i \cdot (s_1 + s_2) + \text{ noise})_{i=1}^n$$

so we can recover $s_1 + s_2$ from the other party's computation even though they do not know what s_1, s_2 are. Note that this is not fully homomorphic encryption because we cannot do multiplication.

2.2 Post Quantum Cryptography

In the above section, we relied on an assumption that SVP is hard in a quantum setting. This seems reasonable for now because we have no quantum algorithms for SVP despite having quantum algorithms for breaking other cornerstones of cryptography like RSA, factoring, and discrete logarithms. Even if quantum computers don't exist yet, there's still pressure to change all the cryptography that we use because maybe they will exist soon.

2.3 Verifying Quantum Computation

When we ask the quantum computer to do something, the quantum computer uses all of the entangled states that perhaps we cannot even write down, and it itself cannot tell us what the entangled states are. However, we want to know if the quantum computer is actually doing the computation we asked it to do. There is a way to verify the quantum computation using LWE [Mah18].

2.4 Public Key Encryption

For concreteness, we show how to use LWE to achieve what is arguably the most fundamental goal of cryptography: public key encryption. To quickly summarize public key cryptography, there's a public key that's shared with everyone, and a corresponding private key that's kept secret. Anyone can use the public key to encrypt a message, but its private key is needed to decrypt a message.

In order to leverage LWE for public key cryptography, we hold onto a private key $s \sim \mathbb{Z}_q^n$. The public key we broadcast to everyone is a set of m samples (a_i, b_i) where $b_i = a_i \cdot s \pmod{q} + \text{noise}$.

The protocol for encrypting one bit of information is then

1. Choose a random subset $S \subseteq [m]$ of the samples.
2. Output $\left(\sum_{i \in S} a_i \ , \ \sum_{i \in S} b_i\right)$ to encode 0, or $\left(\sum_{i \in S} a_i \ , \ \frac{q}{2} + \sum_{i \in S} b_i\right)$ to encode 1.

In order to decrypt a message (a, b), the owner of the secret key can compute $b - a \cdot s$ and compare the result to see if it's closer to 0 (0 bit) or $\frac{q}{2}$ (1 bit).

2.5 Hardness of Learning

Beyond cryptography, hardness of LWE can be viewed as computational impossibility of learning a very simple class of functions (linear functions $\pmod{q}$) in the presence of noise.
We can also use it show hardness of PAC learning other functions. We will focus on *improper* (rather than proper) learning, where the samples are guaranteed to be generated by an hypothesis $h \in \mathcal{H}$ for some class $\mathcal{H}$, but the function that we return doesn't have to come from this class. Last week we proved an intractability result for learning the very simple class of halfspaces in the *agnostic* setting, i.e. we have access to noisy samples. But what about the *realizeable* setting, where we always see the true value of $h(x)$?
In the realizeable setting learning halfspaces is actually easy, but we can use LWE to show that learning a slightly more complicated, namely depth 3 ReLu neural nets, is hard. More generally, there is a natural connection between cryptography and hardness of PAC learning that goes back to Valiant's original paper:

Theorem 2.1 ([Val84], very informal)**.** *If we can do cryptography with $\mathcal{H}$, then no computationally efficient learning is possible for $\mathcal{H}$.*

Intuition: If we can efficiently learn the function that nature used, then we could decrypt messages without using the secret key which breaks the cryptographic security assumption of $\mathcal{H}$.

Thus, we want to find the simplest class of $\mathcal{H}$ that we can do cryptography with so that we can say that even for these simple functions, we cannot learn them.

Theorem 2.2 ([FGKP09])**.** *LWE encryption can be done with a depth-3 ReLU neural network.*

Intuition: The actual LWE encryption function is quite simple since it's just a linear function (takes one layer), but there's also a (mod q) operation which requires a couple of additional layers to express.

Combining theorems 2.1 and 2.2, we see that it is hard to learn depth-3 ReLU neural networks.

References

[Bra18] Zvika Brakerski. Fundamentals of fully homomorphic encryption - A survey. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:125, 2018.

[FGKP09] Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On agnostic learning of parities, monomials, and halfspaces. *SIAM J. Comput.*, 39(2):606–645, 2009.

[Mah18] Urmila Mahadev. Classical verification of quantum computations. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 259–267, 2018.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 333–342, New York, NY, USA, 2009. ACM.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.