4 The Inductive Learning Task

In this chapter, we introduce the problem of acquiring knowledge from examples, i.e., *inductive learning*. We will discuss how formulas that are true in the PAC sense introduced in Chapter 2 can be learned from examples. In Section 4.1, we show that in order to learn, it is sufficient to find the representation of a property that is consistent with our given examples, which we will use to develop algorithms for learning in Sections 4.2–4.4. Continuing the thread from Chapter 3, we consider how the choice of the representations of the learned formula and the breadth of properties we aim to successfully learn, as captured by a family of representations, affects the computational feasibility of the corresponding learning task. In Section 4.5, we show that finding representations that are consistent with our examples is also necessary, which in turn leads to some representations being computationally intractable to learn; but then we see in Section 4.6 that sometimes we can circumvent the intractable problem by learning with a more expressive family of representations. Alas, in Section 4.7 we show that some families of representations cannot be learned, no matter what representation we use. Indeed, we will use these methods to show in Section 4.8 that all but very simple representations are not learnable.

4.1 Occam's razor.

Learning theory was classically concerned primarily with learning definitions from examples. For example, we might wish to learn that a *puppy* is a *dog* that is *young*. We might have many examples of animals, some that are dogs, some that are cats, some that are birds, and so on; and these animals may be young or old; they may be various colors; some may fly; others may speak; some may have long, straight fur and others may have curly fur, and so on. Eventually, we hope our algorithm can infer that all of these attributes except the age and the species are not relevant, and precisely those that are young dogs are called puppies (and perhaps, furthermore that they are not old, not cats nor birds, and so on). In this case, the property to be learned ("puppy") is indicated by a given attribute called the *label* of the example. In the example above, some of the examples are puppies, but many are not.

As a mathematically simple example, let's suppose that the examples are possible worlds given by n Boolean attributes, $x_1, \ldots, x_n$, and that the property c is in fact defined by $x_1 \wedge x_2$. Let's suppose furthermore that the distribution on the possible

worlds is uniformly random, i.e., each $\vec{x} \in \{0,1\}^n$ is observed with probability $1/2^n$. Our task now is to find a representation h such that $h(\vec{x})$ is equivalent to $c(\vec{x})$. But, recall Hume's observation (from Section 2.2) that we cannot hope for h to agree with c for *all* $\vec{x}$; and following Section 2.4, we will only hope to obtain a representation h such that $h(\vec{x}) = c(\vec{x})$ is $1-\epsilon$-valid for the distribution on possible worlds for the examples. This task is known as *supervised learning*.

To perform supervised learning, a natural approach is to search for a representation that simply agrees with the observed examples: find a property that is true when the target property is true, and false when the target property is false. To be precise:

Definition 4.1 The *(proper) consistent representation* problem is the following search problem: for a class of representations C, given $(\vec{x}^{(1)}, c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, c(\vec{x}^{(m)}))$ for some $c \in C$, find $h \in C$ such that $h(\vec{x}^{(i)}) = c(\vec{x}^{(i)})$ for all $i = 1, \ldots, m$. Precisely, the relation is

$$\text{Consis}_C = \left\{ \begin{pmatrix} (\vec{x}^{(1)}, c(\vec{x}^{(1)})) \\ \vdots \\ (\vec{x}^{(m)}, c(\vec{x}^{(m)})) \end{pmatrix}, h \right) : c, h \in C, \text{ and } \forall i = 1, \ldots, m \; h(\vec{x}^{(i)}) = c(\vec{x}^{(i)}) \Bigg\}.$$

If furthermore, each $\vec{x}^{(i)}$ has n attributes, there exists a polynomial s such that h has representation length at most $s(n,m)$, and members of C can be efficiently evaluated, then this is an NP-search problem,

$$\text{Consis}_C^{s(n,m)} = \left\{ \begin{pmatrix} (\vec{x}^{(1)}, c(\vec{x}^{(1)})) \\ \vdots \\ (\vec{x}^{(m)}, c(\vec{x}^{(m)})) \end{pmatrix}, h \right) : \begin{array}{l} c, h \in C, |h| \le s(n,m), \\ \forall i = 1, \ldots, m \; \vec{x}^{(i)} \in \{0,1\}^n \\ \text{and } h(\vec{x}^{(i)}) = c(\vec{x}^{(i)}) \end{array} \Bigg\}.$$

Consis_C is easy to solve in many cases, but does this solve the learning task?

Example 4.1: As a cautionary example, consider the class of database representations. Given our list of labeled examples, $(\vec{x}^{(1)}, c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, c(\vec{x}^{(m)}))$, we let h be the database $\{\vec{x}^{(i)} : (\vec{x}^{(i)}, 1)$ in the given list$\}$. This solves $\text{Consis}_{\text{Database}}$: whenever $c(\vec{x}^{(i)}) = 1$, $\vec{x^{(i)}}$ is in the database, and so $h(\vec{x^{(i)}}) = 1$; conversely, whenever $c(\vec{x}^{(i)}) = 0$, for any j such that $\vec{x}^{(j)} = \vec{x}^{(i)}$, $c(x^{(j)}) = c(x^{(i)}) = 0$, so $x^{(j)}$ is *not* in the database, and thus $h(x^{(j)}) = 0$. We informally say that h *"memorizes"* the examples.
In our example where c is the database $\{\vec{x} \in \{0,1\}^n : x_1 \wedge x_2\}$ for the uniform distribution on possible worlds, however, h generally isn't adequate. Observe, 1/4 of the possible worlds are in c, but if we have only seen m of these, then h is only true for m out of these 2^{n-2} possible worlds. Indeed, if $m < 2^{n-3}$, $h(\vec{x}) \neq c(\vec{x})$ with probability at least $(2^{n-2} - m)/2^n > 1/8$.

Using such a memorization-based approach, we must observe a very large number of examples indeed to be correct with, say, 90% probability. The problem is that the database does not *"generalize"* to correctly classify unseen positive examples, which

occupy a large portion of the possible worlds. Simple memorization is not an efficient strategy for learning.

So, the question is now how we can successfully generalize from a small number of examples to accurate prediction on most future instances. One approach that is prominent in the philosophy of science is known as *Occam's Razor: "entities must not be multiplied without necessity,"* or in other words, among multiple competing theories, choose the simplest one possible. Although "simplicity" here is a fuzzy, informal notion, a straightforward interpretation in our setting would be to take the *shortest* representation that is consistent with the data. In our formal model of learning from independent examples, we can then establish rigorously that Occam's Razor actually leads to correct generalization:

Theorem 4.2 *There is a constant $\kappa > 0$ such that for any class of representations C_n on n attributes and any distribution D on possible worlds for these n attributes, if $s(n, m)$ is a function such that for $\epsilon, \delta > 0$,*

$$s(n,m) \leq \kappa\epsilon m - \log_2 \frac{1}{\delta} - 1$$

$$\Updownarrow$$

$$m \geq \frac{1}{\kappa\epsilon}(s(n,m) + \log_2 \frac{1}{\delta} + 1),$$

then with probability $1 - \delta$, any solution h to $\text{Consis}_{C_n}^{s(n,m)}$ on m examples drawn independently from D labeled by some $c \in C_n$ satisfies

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon.$$

The idea behind the proof of Occam's Razor is very simple: our analysis of surveys from Section 2.3 shows that a single representation that agrees with a sufficient number of independently drawn examples generalizes well (as captured by the ϵ parameter). We only need to worry about the issue that each representation poses a separate "hypothesis test", so we need to address the issue that the likelihood of accepting a representation that does not generalize (captured by the δ parameter) grows with the number of representations we consider. But, the restriction on the length of the representations $s(n, m)$ limits the number of representations under consideration. We only need to scale down the likelihood of each test failing by the number of possible representations, so that the overall probability of a bad representation remaining a candidate for our final representation remains below the desired tolerance δ.

Proof For brevity, let us say that a representation $h \in C_n$ of length $s(n, m)$ is "bad" if $\Pr_{\vec{x} \in D}[h(\vec{x}) \neq c(\vec{x})] > \epsilon$, i.e., if h remaining a solution to $\text{Consis}_{C_n}^{s(n,m)}$ would violate the conclusion. For any bad h, observe that h is only consistent with a possible world drawn from D labeled by c with probability less than $1 - \epsilon$; thus, the probability that it is consistent with m such examples drawn independently from D is less than $(1 - \epsilon)^m$. Recalling $1 - x \leq e^{-x}$ (where e is the base of the natural logarithm), we observe that $(1 - \epsilon)^m \leq e^{-\epsilon m}$. Thus, for m, $s(n, m)$, ϵ, and δ satisfying the conditions of the

theorem, this probability is at most

$$e^{-\epsilon \cdot \frac{1}{\kappa\epsilon}\left(s(n,m)+\log_2 \frac{1}{\delta}+1\right)}.$$

Thus, observe that for $\kappa = \frac{1}{\ln 2}$, the probability is at most

$$e^{-\ln 2\left(s(n,m)+\log_2 \frac{1}{\delta}+1\right)} = \frac{1}{2^{s(n,m)+1}} \cdot \delta.$$

We know that the number of representations that can be represented using at most $s(n,m)$ bits is $\sum_{\ell=0}^{s(n,m)} 2^\ell = 2^{s(n,m)+1} - 1$.[1] Since the bad representations must be of length at most $s(n,m)$, we can take a union bound over the fewer than $2^{s(n,m)+1}$ events that each such bad h remains a candidate solution, to obtain that the overall probability that a bad candidate remains is at most $2^{s(n,m)+1} \cdot \frac{1}{2^{s(n,m)+1}} \cdot \delta = \delta$ as needed: with probability $1-\delta$, now, all solutions to $\text{Consis}_{C_n}^{s(n,m)}$ are not bad. □

Occam's Razor is adequate for the analysis of learning with the kinds of discrete representations we consider here. We note that it is needlessly pessimistic in its treatment of numerical representations, for which an analysis based on the "VC-dimension" will yield sharper results: see Appendix A. Regardless of whether we use Occam's Razor or the VC-dimension bounds, we observe that the number of examples sufficient to guarantee generalization grows with the complexity of consistent representations.

In particular, recalling Theorem 3.15, for any family of representations, most properties will require (exponentially) large representations. In such cases, the bound given by Occam's Razor is essentially no better than the number of examples needed to observe and memorize the truth table on (almost) all possible worlds. Thus, efficient learning relies on a "representational bias" in which the property of interest has a short representation. Although some representations, such as truth tables, are inherently useless for efficient learning, most implicitly select out a small set of efficiently learnable properties. Contrapositively, we stress that whether or not the representations are "universal" (i.e., can represent all possible properties) is thus all but irrelevant to learning, since all but a vanishingly small fraction of the possible properties require an intractably large number of examples to learn.

4.2 Learning conjunctions via Elimination.

We now know that representations that are consistent with a sufficiently large number of examples relative to their length will give accurate predictions in the future. But, clearly, simply listing out the examples we've seen so far generally cannot give us such a short representation. For most families C and reasonable size bounds s, $\text{Consis}_{C_n}^{s(n,m)}$ is not such a trivial computational problem; we now turn to the problem of finding algorithms to solve such problems.

[1] Indeed, the left-hand-side, in binary, is $\overbrace{11\cdots1}^{s(n,m)}$ and $\overbrace{11\cdots1}^{s(n,m)}+1 = 1\overbrace{00\cdots0}^{s(n,m)}$ in binary.

A simple first example of such an algorithm is for the class of *conjunctions*, introduced in Section 3.2, which are given by an AND of literals. The algorithm, which appeared as far back as in Mill's philosophy, is very simple and given in Algorithm 1.

Input: Examples $((\vec{x}^{(1)}, c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, c(\vec{x}^{(m)}))$.
Output: A conjunction h solving $\text{Consis}^{2n}_{\text{Conjunction}_n}$.
begin
  Initialize h to contain all $2n$ literals on the n attributes.
  `// Ensure positive examples are labeled correctly`
  for $i = 1, \ldots, m$ **do if** $c(\vec{x}^{(i)}) = 1$ **then**
    forall $\ell \in h$ **do if** $\ell(\vec{x}^{(i)}) = 0$ **then** Remove ℓ from h.

  `// Check negative examples are labeled correctly`
  for $i = 1, \ldots, m$ **do if** $c(\vec{x}^{(i)}) = 0$ **then**
    Put $caught \leftarrow 0$
    forall $\ell \in h$ **do if** $\ell(\vec{x}^{(i)}) = 0$ **then** $caught \leftarrow 1$.

    if $caught = 0$ **then return** $\perp$.

  return h.
end

Algorithm 1: Elimination Algorithm for learning a conjunction

Theorem 4.3 *The Elimination Algorithm solves* $\text{Consis}^{2n}_{\text{Conjunction}_n}$ *in time* $O(mn)$.

Proof To begin, note that if we represent our conjunctions by a vector of $2n$ bits, one per literal indicating whether it is present in h or not, we indeed satisfy the length bound of $2n$ bits for $\text{Consis}^{2n}_{\text{Conjunction}_n}$.

Note that at the end of the first loop, h does not contain any literal ℓ such that on some given possible world $\vec{x}^{(i)}$ for which $c(\vec{x}^{(i)})$ is true, $\ell(\vec{x}^{(i)})$ is false. Therefore, h is true on each such $\vec{x}^{(i)}$.

Now, suppose that the input examples indeed pair each given possible world $\vec{x}^{(i)}$ with $c(\vec{x}^{(i)})$ for some conjunction c. Observe that any literal ℓ that is false on some $\vec{x}^{(i)}$ where $c(\vec{x}^{(i)})$ is true cannot be in c. Therefore, at the end of the first loop, h contains all of the literals of c. Now, during the second loop, whenever $c(\vec{x}^{(i)})$ is false, there must exist some ℓ^* in c for which $\ell^*(\vec{x}^{(i)})$ is false. Since h contains all of the literals of c, h contains ℓ^* in particular, and so $h(\vec{x}^{(i)})$ is also false, and moreover $caught$ is set to 1 when ℓ^* is considered. The algorithm therefore returns h, which indeed agrees with all of the given examples.

Of course, if no conjunction is consistent with all of the given examples, then in particular the conjunction h obtained at the end of the first loop – which is necessarily consistent with all of the examples labeled true – must be inconsistent with some

example $\vec{x}^{(i)}$ labeled false. In this case, we note that the algorithm indeed returns $\perp$ when this example is considered.

Finally, the running time is immediate: the outer loops run for m iterations and the inner loops run for at most $2n$ iterations of $O(1)$ time each, giving $O(mn)$ time overall. □

It now follows from Occam's Razor that Elimination solves the supervised learning problem for conjunctions:

Corollary 4.4 *For any distribution D on possible worlds on n attributes, given*

$$m \geq \frac{\ln 2}{\epsilon}(2n + \log_2 \frac{1}{\delta} + 1)$$

possible worlds drawn independently from D labeled by a conjunction c, the Elimination Algorithm returns a conjunction h such that with probability $1 - \delta$,

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon.$$

The Elimination Algorithm has one additional useful property: if the property of interest is actually represented by a conjunction, then the representation returned by the Elimination Algorithm is always correct when it predicts the property holds. As a consequence, the algorithm is moreover still useful when given only the positive examples.

Corollary 4.5 *Given possible worlds labeled by a conjunction c as input, the Elimination Algorithm returns a conjunction h such that if h holds for a possible world $\vec{x}$, then c also holds for $\vec{x}$.*

Proof We noted that h must contain all of the literals of c, and so whenever c is false on a possible world $\vec{x}$, any literal ℓ^* that is false in c is also contained in h. So, h is also false on $\vec{x}$. Contrapositively, if h is true of $\vec{x}$, so is c. □

We can illustrate the value of this property by returning to the example of the box-moving robot, or more generally STRIPS environments as introduced in Section 2.4.

Example 4.2: Recall that in STRIPS environments, the actions can only be executed when an associated *precondition* holds. These preconditions are conjunctions of literals on the fluents. For example, to execute the action, "pick up a box in room 1," both the robot and the box must be in room 1.

Let's suppose we would like to learn the preconditions of the actions in the environment by observing the robot. But if the robot is only executing correct plans, we will never observe an example of the robot attempting to take the action and failing when the precondition is false. The precondition that is always true is then necessarily consistent with the observations, but clearly incorrect if the actual precondition is nontrivial: attempting to create plans for the robot using these learned preconditions could lead to the creation of plans that fail.

The property of the Elimination Algorithm that it cannot make false positive predictions now addresses this problem: if we use Elimination to learn a conjunction h from the examples where the action was observed, then whenever h predicts that the precondition holds, the action actually can be taken. We also obtain that after $O(\frac{1}{\epsilon}(n + \log \frac{1}{\delta}))$ examples (with n fluents), with probability $1 - \delta$ over the examples we've seen, the precondition will correctly predict that the action can be taken with probability at least $1 - \epsilon$ in the distribution on worlds of future examples.

4.3 Reducing disjunction learning to conjunction learning.

Our first nontrivial learning algorithm, the Elimination Algorithm, efficiently learns conjunctions. Conjunctions are just as simple as disjunctions – the AND is replaced by an OR – and it is easy to modify the Elimination Algorithm to learn disjunctions instead, and analyze its efficiency and correctness by analogous arguments. But, it is also possible to reuse the Elimination Algorithm for conjunctions as-is, along with its analysis, by instead *transforming* the disjunction learning problem into a conjunction learning problem—i.e., by a *reduction* between the learning problems. We saw reductions between relation problems in Section 3.6. Reductions between learning problems are essentially analogous, except that the definition of a learning problem is more complex: Learning problems, in contrast to the worst-case relation problems we defined previously, include a probability distribution on possible worlds. We will postpone the formal definitions capturing such problems, and hence reductions in general, until they become necessary. Presently, we'll show concretely how to reduce the learning of disjunctions to conjunction learning, as solved by the Elimination Algorithm.

The key observation underlying the reduction is De Morgan's Law (Proposition 3.5): given a disjunction $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$, we can rewrite the disjunction as a conjunction by introducing a double-negation and invoking De Morgan's law:

$$\begin{aligned}\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k &= \neg\neg\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k \\ &= \neg[(\neg\ell_1) \wedge (\neg\ell_2) \wedge \cdots \wedge (\neg\ell_k)]\end{aligned}$$

Succinctly, for the disjunction $c(\vec{x}) = \bigvee_{i=1}^{k} \ell_i$ and the conjunction $\tilde{c}(\vec{x}) = \bigwedge_{i=1}^{k} \neg\ell_i$, we have $\neg c(\vec{x}) = \tilde{c}(\vec{x})$. So, here is our algorithm: we are given examples $(\vec{x}^{(1)}, c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, c(\vec{x}^{(m)}))$ where each $\vec{x}^{(i)}$ is drawn independently from the same probability distribution D on possible worlds. We pass the examples $(\vec{x}^{(1)}, \neg c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, \neg c(\vec{x}^{(m)}))$ to the Elimination Algorithm. The Elimination Algorithm returns a conjunction $\tilde{h}$. We negate each literal in $\tilde{h}$, and return a disjunction with this set of negated literals, h. This algorithm runs in linear time in the input size, plus the time taken by the Elimination Algorithm (which, it turns out, is also linear time in the input size).

To see that the algorithm is correct, observe that the examples $(\vec{x}^{(1)}, \neg c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, \neg c(\vec{x}^{(m)}))$ constructed by the reduction are equivalent to examples $(\vec{x}^{(1)}, \tilde{c}(\vec{x}^{(1)}))$,

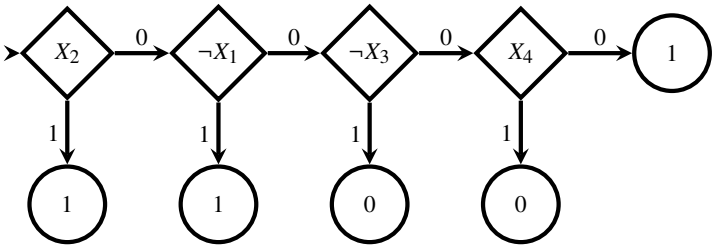


Figure 4.1 A decision list. 1000 evaluates to 0, but 0010 evaluates to 1, as does 1010.

$\ldots, (\vec{x}^{(m)}, \tilde{c}(\vec{x}^{(m)}))$—i.e., i.i.d. examples labeled by the conjunction $\tilde{c}$. Our analysis of the Elimination Algorithm thus guarantees that the conjunction $\tilde{h}$ it returns obeys

$$\Pr_{\vec{x} \in D}[\tilde{h}(\vec{x}) = \tilde{c}(\vec{x})] \geq 1 - \epsilon$$

with probability $1 - \delta$, given m is sufficiently large, $m \geq \frac{\ln 2}{\epsilon}(2n + \log_2 \frac{1}{\delta} + 1)$. But by De Morgan's law, $h(\vec{x}) = \neg\tilde{h}(\vec{x})$ and $c(\vec{x}) = \neg\tilde{c}(\vec{x})$. Thus, we also have

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon$$

with probability $1 - \delta$. Thus, h solves the disjunction learning problem.

4.4 Learning decision lists.

We next give an algorithm for learning a more sophisticated representation, *decision lists*:

Definition 4.6 A *decision list* is a decision tree in which at least one child of each internal node is a leaf.

Equivalently, a decision list is given by an ordered list of literals $\ell_1, \ldots, \ell_k$ for some integer k with associated Boolean labels $b_1, \ldots, b_k \in \{0, 1\}$ and a "tail" label $b_{k+1} \in \{0, 1\}$, with the interpretation that we sequentially evaluate $\ell_1(\vec{x})$, $\ell_2(\vec{x})$, and so on until some $\ell_i(\vec{x})$ is true, at which point the decision list returns the associated label b_i, or until all k literals have been tested false, at which point the decision list returns the tail label b_{k+1}.

An example of a decision list is given in Figure 4.1.

We'll use Occam's Razor, and will approach the problem by giving an algorithm to solve the problem of finding a decision list of at most n nodes—note that along the lines of Proposition 3.9, the decision list only needs to read each variable at most once. We can find such a consistent decision list by a simple greedy algorithm. The algorithm repeatedly searches for a *"useful"* literal:

Definition 4.7 We say that a literal ℓ is *useful* for a set of possible worlds S labeled by a property c if $\ell(\vec{x})$ is true of at least one $\vec{x} \in S$ and either for all $\vec{x} \in S$ $c(\vec{x})$ holds, or for all $\vec{x} \in S$, $c(\vec{x})$ does not hold. In the first case, we say it is useful *with label* 1, and in the second case it is useful *with label* 0.

Similar algorithms have been proposed for decision trees more generally, but in contrast to the special case of decision lists, these algorithms for general decision trees do not guarantee successful generalization for general distributions.

Input: List of examples $S = ((\vec{x}^{(1)}, c(\vec{x}^{(1)})), \ldots, (\vec{x}^{(m)}, c(\vec{x}^{(m)}))$.
Output: A decision list h solving $\text{Consis}^{n(\log n+2)}_{\text{DecisionList}_n}$.
begin
  Initialize h to the empty decision list.
  while *Examples remain in S with c true and c false* **do**
    forall *literals* ℓ **do if** ℓ *is useful for S (with label b)* **then**
      Append ℓ with the label b to the end of h.
      Set $S \leftarrow S \setminus \{\vec{x} \in S : \ell(\vec{x}) = 1\}$
  end
  if *All examples in S are labeled* 1 **then** Append tail label 1 to h
  else Append tail label 0 to h
  return h.
end

Algorithm 2: Greedy algorithm for learning a decision list

Theorem 4.8 *Algorithm 2 solves* $\text{Consis}^{n(\log n+2)}_{\text{DecisionList}_n}$ *in time* $O(mn^2)$.

Proof We first note that no literal can be identified as useful twice since we remove all examples from S that are satisfied by the literal when we encounter it. Indeed, once each variable has appeared in a literal, since all remaining examples would satisfy the negation of any of these literals, at most n literals appear in the list. Thus, since each literal can be represented by $\log n + 1$ bits and the label is one bit, we can represent the output h by $n(\log n + 2)$ bits.

Next, we show that in every iteration of the loop, a useful literal exists. Indeed, we are supposing that there exists a decision list c such that the labels are given by c. We note that at least one example remaining in S must satisfy a literal of c, or else only examples that reach the tail of c would remain; since these all receive the tail label, the loop would terminate. Consider the first literal ℓ^* in c such that some example remaining in S satisfies ℓ^*. Observe that any of the remaining examples in S that satisfy ℓ^* must also receive the label associated with ℓ^*, since none of them satisfy any of the earlier literals of c. Thus, ℓ^* is a useful literal for S.

Now, we note that the decision list we construct is consistent with the input examples: by induction on the iterations of the main loop, after the ith iteration, all examples labeled by the decision list are labeled correctly. Initially, no examples are labeled, and since all of the examples removed from S by the ith iteration will already be labeled (correctly) by h, we observe that appending a useful literal to S correctly labels all of the examples satisfied by that literal. This establishes the induction hypothesis for $i + 1$. Finally, when the remaining examples all possess the same label (after the final iteration), they are all indeed correctly labeled if that label is appended as the tail

of the list. Since we argued above that the list only needs to include n literals, the loop terminates after at most n iterations and thus the algorithm returns a decision list that agrees with the given labels, hence solving $\text{Consis}^{n(\log n+2)}_{\text{DecisionList}_n}$.

Since each iteration of the loop can be easily implemented in $O(mn)$ time, the $O(mn^2)$ time bound is immediate. □

And hence, by Occam's Razor,

Corollary 4.9 *For any distribution D on possible worlds on n attributes, given*

$$m \geq \frac{\ln 2}{\epsilon}(n(\log_2 n + 2) + \log_2 \frac{1}{\delta} + 1)$$

possible worlds drawn independently from D labeled by a decision list c, Algorithm 2 returns a decision list h such that with probability $1 - \delta$,

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon.$$

4.5 Representation-specific hardness of learning three-term DNF.

A superficial reading of Occam's Razor might seem to suggest that any property we can efficiently represent can be learned. But, Occam's Razor only addresses the number of examples that must be observed. It does not consider the computational complexity of the problem of finding a short consistent representation. It turns out that this problem may be intractable, even for some seemingly simple representations.

Before continuing on this line of reasoning, we first stop to ask if the problem of finding a consistent representation is really necessary for learning – that is, is there any kind of a converse to Occam's Razor? Or is there some other way of learning?

To answer such questions, we need to formally define what we mean by efficient learning. A natural first definition that captures the guarantees we've given so far is the following:

Definition 4.10 *(Proper) PAC-learning* of the representation class C_n in time $t(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ is the following task: we suppose there exists a probability distribution D on possible worlds with n attributes, and we are given parameters $\epsilon, \delta > 0$ and $t(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ examples $\vec{x}^{(i)}$ drawn independently from D and labeled by some $c \in C$. The algorithm A must run in time $t(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ and with probability $1 - \delta$ return $h \in C$ such that

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon.$$

If A only uses $m(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ examples for some function m, we say A learns with *sample complexity* $m(n, \frac{1}{\epsilon}, \frac{1}{\delta})$.

We now show that to meet this definition for a polynomial t for any class of representations that can be evaluated in polynomial time, it is indeed necessary to solve $\text{Consis}^{p(n,m)}_{C_n}$ for some polynomial p in polynomial time.

Theorem 4.11 *Suppose that there is an algorithm A for proper PAC-learning of the class C_n in time $t(n, \frac{1}{\epsilon}, \frac{1}{\delta})$. Suppose that the evaluation problem for size s members of C_n can be solved in time $t'(s, n)$. Then there is a randomized algorithm that, given as input a list of m examples on n attributes labeled by some $c \in C_n$ runs in time $O(t(n, 2m, 2) + mt'(t(n, 2m, 2), n))$, with probability at least $1/2$ returns a solution to $\text{Consis}_{C_n}^{t(n,2m,2)}$ for the input list, and otherwise outputs $\perp$, indicating failure.*

In particular, since polynomials are closed under sums, products, and compositions, $t(n, 2m, 2) + mt'(t(n, 2m, 2), n)$ is also a polynomial in n and m. Note that the probability of the randomized algorithm succeeding can be amplified to any desired $1 - \delta$ by running it $\log_2 \frac{1}{\delta}$ times. The probability that all of these independent trials fail is at most $(1 - 1/2)^{\log_2 \frac{1}{\delta}} = \delta$.

Proof Here is our algorithm: we construct a new list of examples by sampling with replacement from the original list $t(n, 2m, 2)$ times, running the algorithm A on this list with parameters $\epsilon = 1/2m$ and $\delta = 1/2$, and checking if the output h agrees with the labels for the input list. If so, the algorithm returns h and otherwise it returns $\perp$.

We first note that because A runs in time $t(n, 2m, 2)$, it cannot output h of size greater than $t(n, 2m, 2)$. Now, from the construction of our new algorithm, since evaluating h on each of the m examples can be performed in time $t'(t(n, 2m, 2), n)$ it is clear that our new algorithm runs in time $O(t(n, 2m, 2) + mt'(t(n, 2m, 2), n))$, and also either returns h solving $\text{Consis}_{C_n}^{t(n,2m,2)}$ for the input list or $\perp$.

Now, to see that the new algorithm returns the solution to $\text{Consis}_{C_n}^{t(n,2m,2)}$ with probability at least $1/2$, we suppose that the examples are indeed labeled by a member of the class C_n. Let D denote the uniform distribution on the input list of possible worlds, and note that the input our new algorithm provides to A is sampled from this distribution. Thus, since we are given that A is a proper PAC-learning algorithm, with probability at least $1/2$ it returns h such that $\Pr_{\vec{x} \in D}[c(\vec{x}) = h(\vec{x})] \geq 1 - \frac{1}{2m}$. But observe, since D is uniform over the m inputs, if $c(\vec{x}^{(i)}) \neq h(\vec{x}^{(i)})$ for any i, we would have

$$\Pr_{\vec{x} \in D}[c(\vec{x}) = h(\vec{x})] \leq 1 - \frac{1}{m} < 1 - \frac{1}{2m}.$$

Therefore, we must have that with probability at least $1/2$, $c(\vec{x}^{(i)}) = h(\vec{x}^{(i)})$ for all $i = 1, \ldots, m$, so h indeed is a solution to $\text{Consis}_{C_n}^{t(n,2m,2)}$ with probability at least $1/2$. □

We note that the bound on the size of the representation we obtain is not sufficient to apply Occam's Razor and obtain generalization – indeed, for example, we can always obtain a DNF of size $O(mn)$ by just converting a database of the positive examples to the DNF representation, but this clearly will not generalize in many cases. Nevertheless, it shows that it must be possible to solve the consistency problem for *some polynomial size* in *some polynomial time*, and even this somewhat relaxed problem turns out to be intractable in many cases.

A simple example occurs for the representation class of DNFs with *three* terms. That is, formulas of the form $T_1 \vee T_2 \vee T_3$ where each of T_1, T_2, and T_3 is an AND of

literals that we refer to as a term in this context. Since each term may be represented by a block of $2n$ bits indicating for each literal whether or not it is present in the term, we can represent any three-term DNF on n attributes using $6n$ bits. Thus, if *any* three-term DNF exists that is consistent with a set of example possible worlds, it gives a solution to $\text{Consis}^{6n}_{3-\text{TermDNF}_n}$. We'll show that $\text{Consis}^{6n}_{3-\text{TermDNF}_n}$ is NP-complete. By Theorem 4.11, this will imply that any algorithm for proper PAC-learning of three-term DNFs would give randomized polynomial time algorithms for all NP search problems. Because it is widely believed that some NP search problems do not have polynomial time algorithms even when the algorithms are randomized, we view this as evidence that no efficient algorithm exists for proper PAC-learning of three-term DNFs.

To show that a problem is NP-complete, we typically proceed by presenting a reduction from a problem that is known to be NP-complete. In this case, we will reduce from the 3-coloring problem: given an undirected graph $G = (V, E)$, a *proper 3-coloring* of G is an assignment of one of three possible *colors* to each vertex of G, such that any pair of vertices u and v joined by an edge in G receive distinct colors. Formally, if $\chi : V \to \{R, Y, B\}$ is a function describing such an assignment, we have for all $\{u, v\} \in E$ that $\chi(u) \neq \chi(v)$. The 3-coloring search problem is to find a proper coloring of a given graph. 3-coloring is NP-complete.

Such a reduction converts a graph G into a set of labeled examples $\{(\vec{x}^{(i)}, b^{)(i)}) : i = 1, \ldots, m\}$ such that we can "read off" a proper 3-coloring of G from any three-term DNF that is consistent with the examples. The idea, naturally, is to interpret the terms as assigning colors. Since terms are sets of literals, we therefore wish to interpret the literals as vertices. We need our examples to enforce that no adjacent pair of vertices obtains the same color; that is, if our scheme for encoding colors into terms would assign such a color to adjacent vertices, one of the input examples must not receive its given label. The one trick here is to interpret the color assignment being given by which terms are satisfied on the positive examples, which will correspond to the *absence* of some corresponding literals from those terms. We'll give the full construction in the proof of the theorem below.

Theorem 4.12 *If there is a polynomial time algorithm for proper PAC-learning of three-term DNF, then every NP search problem has a randomized polynomial time algorithm.*

Proof Since each three-term DNF can be represented in $6n$ bits, Theorem 4.11 shows that any polynomial-time algorithm for proper PAC-learning of three-term DNF gives a randomized polynomial-time algorithm for $\text{Consis}^{6n}_{3-\text{TermDNF}_n}$. We will reduce 3-coloring to $\text{Consis}^{6n}_{3-\text{TermDNF}_n}$; since 3-coloring is NP-complete and polynomial-time reductions can be composed, this would give a randomized polynomial-time algorithm for every NP-search problem.

The reduction is as follows: we are given an undirected graph $G = (V, E)$. We create the following set of $|V| + |E|$ examples on $|V|$ attributes. For each ith vertex,

$i = 1, \ldots, |V|$, we create a positive example:

$$((1, 1, \ldots, 1, \overbrace{0}^{x_i}, 1, \ldots, 1), 1).$$

For each edge $\{i, j\} \in E$, we create a negative example:

$$((1, 1, \ldots, 1, \overbrace{0}^{x_i}, 1, \ldots, 1, \overbrace{0}^{x_j}, 1, \ldots, 1), 0).$$

This is our instance for $\text{Consis}^{6|V|}_{3-\text{TermDNF}_{|V|}}$. Given a solution, $T_1 \vee T_2 \vee T_3$, we obtain a coloring χ as follows: for the ith vertex, if the positive example $(1, 1, \ldots, 1, \overbrace{0}^{x_i}, 1, \ldots, 1)$ satisfies T_1, we color the vertex R; or, if it satisfies T_2 but not T_1, we color it Y; or, if it does not satisfy T_1 or T_2, it must satisfy T_3 (since the example is labeled 1), and we color it B. We thus obtain a coloring of G, that we return. Observe that as long as we solve $\text{Consis}^{6|V|}_{3-\text{TermDNF}_{|V|}}$ in polynomial time, we obtain our coloring in polynomial time (using our possibly randomized algorithm).

To establish the reduction is correct, we need to show two things: First, that if G has a proper 3-coloring, there is a solution to $\text{Consis}^{6|V|}_{3-\text{TermDNF}_{|V|}}$. And, second, that our reduction produces proper 3-colorings.

For the first part, we suppose that we have some coloring χ^*. We then consider the following three-term DNF: let T_1 contain the literals x_i for which $\chi^*(v_i) \neq R$ (any color but R); similarly, let T_2 contain the literals x_i for which $\chi^*(v_i) \neq Y$, and T_3 contain x_i for which $\chi^*(v_i) \neq B$. We first observe that the positive examples are labeled correctly, since each vertex i receives some color C, and then the corresponding term T_c contains only positive literals for a set of attributes *not* corresponding to the ith vertex. Thus, $(1, 1, \ldots, 1, \overbrace{0}^{x_i}, 1, \ldots, 1)$ indeed satisfies T_c, and thus our DNF is satisfied. Now, we see that the negative examples are labeled correctly as well: indeed, for $\{v_i, v_j\} \in E$, since the 3-coloring is proper, $\chi^*(v_i) = C$ and $\chi^*(v_j) = C'$ for $C \neq C'$. The term corresponding to C contains x_j (since v_j *does not* obtain color C), the term corresponding to C' similarly contains x_i, and the term corresponding to the third color contains both x_i and x_j. Thus, all three terms have at least one falsified literal, and hence the DNF is falsified, as needed.

Finally, we show that any coloring the reduction obtains is necessarily proper. We've already seen that the reduction assigns a color to each vertex. We only need to now show that the colors assigned to vertices joined by an edge are distinct. We will argue by the contrapositive: if two vertices receive the same color, they cannot be joined by an edge. Indeed, consider v_i and v_j such that $\chi(v_i) = \chi(v_j)$. We see that the algorithm produces such a coloring only if the same term T_c was satisfied on the positive examples

$$(1, 1, \ldots, 1, \overbrace{0}^{x_i}, 1, \ldots, 1) \text{ and } (1, 1, \ldots, 1, \overbrace{0}^{x_j}, 1, \ldots, 1).$$

Observe that this term cannot contain $x_i, x_j, \neg x_i$, or $\neg x_j$, or else it would be falsified in one of these examples. But now, if there were an edge $\{v_i, v_j\} \in E$, we would find

that T_c is also satisfied on the *negative* example for the edge,

$$((1,1,\ldots,1,\overbrace{0}^{x_i},1,\ldots,1,\overbrace{0}^{x_j},1,\ldots,1),0).$$

since it only differs from the positive examples in the values taken by x_i and x_j, neither of which may appear in T_c. The DNF would then incorrectly label the example 1, and thus would *not* be a solution to $\text{Consis}^{6|V|}_{3-\text{TermDNF}_{|V|}}$. Therefore, any solution must assign distinct colors to adjacent vertices, and hence must be proper, as needed. □

4.6 Improper learning of three-term DNF by 3CNF.

Although proper PAC-learning of three-term DNF seems to be intractable, *strictly* more expressive representations *do* have efficient proper PAC-learning algorithms. In particular, recall that 3CNF is the family of formulas given by an AND of clauses, each of which is an OR of at most three literals. 3CNFs are (strictly[2]) more expressive than three-term DNFs:

Proposition 4.13 *Every property that is represented by a three-term DNF is also represented by a 3CNF.*

Proof Observe that AND and OR satisfy the following distributive law:

$$(A \wedge B) \vee C \Leftrightarrow (A \wedge C) \vee (B \wedge C).$$

So we can rewrite the three term DNF $T_1 \vee T_2 \vee T_3$ as follows. Let $T_1 = \ell_{1,1} \wedge \ell_{1,2} \wedge \cdots \wedge \ell_{1,k}$. Interpreting $A = \ell_{1,1}$, $B = \ell_{1,2} \wedge \cdots \wedge \ell_{1,k}$, and $C = T_2 \vee T_3$, we get that $T_1 \vee T_2 \vee T_3$ may be rewritten as

$$(\ell_{1,1} \vee T_2 \vee T_3) \wedge ([\ell_{1,2} \wedge \cdots \wedge \ell_{1,k}] \vee T_2 \vee T_3).$$

Iterating k times, we thus obtain:

$$(\ell_{1,1} \vee T_2 \vee T_3) \wedge \cdots \wedge (\ell_{1,k} \vee T_2 \vee T_3).$$

Now we similarly denote $T_2 = \ell_{2,1} \wedge \cdots \wedge \ell_{2,m}$ and $T_3 = \ell_{3,1} \wedge \cdots \wedge \ell_{3,n}$. Interpreting, e.g., $A = \ell_{2,1}$, $B = \ell_{2,2} \wedge \cdots \wedge \ell_{2,m}$, and $C = \ell_{1,1} \vee T_3$, we can rewrite the subformula $\ell_{1,1} \vee T_2 \vee T_3$ as

$$(\ell_{1,1} \vee \ell_{2,1} \vee T_3) \wedge (\ell_{1,1} \vee [\ell_{2,2} \wedge \cdots \wedge \ell_{2,m}] \vee T_3)$$

and thus iterating similarly for $j = 1,\ldots,m$ for $i = 1,\ldots,k$,

$$\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{m} (\ell_{1,i} \vee \ell_{2,j} \vee T_3)$$

and then finally we similarly expand T_3. □

[2] Strictness is witnessed by $\neg Tribes$, see Exercise 3.2, but this isn't important here.

Like three-term DNFs, 3CNFs cannot be too large: observe that there are only $2^3\binom{n}{3} = 8 \cdot \frac{n(n-1)(n-2)}{6}$ distinct clauses of three literals on n Boolean attributes; similarly, there are $2n$ literals (clauses of size one) and $4\frac{n(n-1)}{2}$ clauses of two literals. Thus, there are at most $O(n^3)$ clauses that may appear in any such formula; hence these formulas can also be represented by $O(n^3)$ bits indicating which of the clauses appear and which are absent. We can learn 3CNFs by a reduction to conjunctions, using the Elimination algorithm:

Theorem 4.14 *There is a proper PAC-learning algorithm for 3CNF running in time $O(mn^3)$ with sample complexity $O(\frac{1}{\epsilon}(n^3 + \log\frac{1}{\delta}))$.*

Proof As promised, we will reduce learning 3CNF to learning conjunctions. We suppose we have a set of labeled possible worlds with n Boolean attributes $x_1, \ldots, x_n$. We will construct a new set of attributes as follows: for each clause C on at most three literals, we create an attribute y_C; we will map each input example $\vec{x}^{(i)}$ to a possible world $\vec{y}^{(i)}$ on these new attributes where $y_C^{(i)} = C(\vec{x}^{(i)})$ (i.e., the evaluation of C on $\vec{x}^{(i)}$). We give $y^{(i)}$ the same label as the original ith example. Since each clause may be evaluated in time $O(1)$ (it only mentions $O(1)$ attributes) we can construct these examples in time $O(mn^3)$. We run Elimination on this new set of examples to obtain a conjunction $\tilde{h}$ over the attributes $\vec{y}$. We then map the conjunction back to a 3CNF h: if the conjunction contains the literal y_C, we include the clause C in h, and if it contains $\neg y_C$ for $C = \ell_1 \vee \ell_2 \vee \ell_3$, we include the three clauses $\neg\ell_1$, $\neg\ell_2$, and $\neg\ell_3$ in h. Observe that this is a 3CNF such that for any $\vec{y}$ in the image of our map for $\vec{x}$, $h(\vec{x}) = \tilde{h}(\vec{y})$.

For correctness, we first observe that when the input is labeled by some 3CNF $c^*(\vec{x})$, there exists a conjunction $\tilde{c}^*(\vec{y})$. Indeed, consider the conjunction $\tilde{c}^*$ that contains precisely the literals y_C for clauses C that appear in c^*. Then since we put $y_C^{(i)} = C(\vec{x}^{(i)})$ for each C, it is immediate that $\tilde{c}^*(\vec{y}^{(i)}) = c^*(\vec{x}^{(i)})$. Elimination then runs in time $O(mn^3)$ (since there are now $O(n^3)$ attributes) with sample complexity $O(\frac{1}{\epsilon}(n^3 + \log\frac{1}{\delta}))$ and returns, with probability $1 - \delta$, a conjunction $\tilde{h}$ on $\vec{y}$ such that

$$\Pr_{\vec{y}\in\tilde{D}}[\tilde{h}(\vec{y}) = \tilde{c}^*(\vec{y})] \geq 1 - \epsilon$$

where $\tilde{D}$ is the distribution of the image of our map on the original distribution D on possible worlds. But now, since we had $h(\vec{x}) = \tilde{h}(\vec{y})$ by construction as well, we therefore also have

$$\Pr_{\vec{x}\in D}[h(\vec{x}) = c^*(\vec{x})] \geq 1 - \epsilon$$

and thus our algorithm is a proper PAC-learning algorithm for 3CNF with sample complexity $O(\frac{1}{\epsilon}(n^3 + \log\frac{1}{\delta}))$ and overall running time $O(mn^3)$ as claimed. □

Now, because three-term DNFs can be represented by 3CNFs and 3CNFs can be efficiently learned, we will say that three-term DNFs can be *improperly* learned using the representation of 3CNFs. Observe that for the purposes of predicting labels, the 3CNF representation is at most polynomially larger ($n \mapsto O(n^3)$) and can be evaluated in linear time in its representation size, thus polynomial time in terms of the original

formula. So, these 3CNFs are generally adequate for labeling possible worlds. In general:

Definition 4.15 Let C_n and $\mathcal{H}_n$ be classes of representations on the same n attributes. We say that C_n is *improperly PAC-learnable* using $\mathcal{H}_n$ in time $t(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ and sample complexity $m(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ if there is an algorithm such that for any m examples drawn independently from a distribution D on n attribute possible worlds and labeled by some $c \in C_n$, given these examples and the parameters ϵ and δ, with probability at least $1-\delta$ outputs a representation $h \in \mathcal{H}_n$ such that

$$\Pr_{\vec{x} \in D}[h(\vec{x}) = c(\vec{x})] \geq 1 - \epsilon.$$

If C_n is improperly PAC-learnable in polynomial time and sample complexity by any class $\mathcal{H}_n$ for which evaluation can be computed in polynomial time, then we simply say that C_n is *efficiently (improperly) PAC-learnable*.

Note that efficient (improper) PAC-learnability translates "downward" to less-expressive classes of representations: precisely, if C can be expressed by another representation C' that is efficiently PAC-learnable (by $\mathcal{H}$), then C is also efficiently PAC-learnable by $\mathcal{H}$. Unlike proper PAC-learning, improper PAC-learning only becomes easier as we consider less-expressive representations. Conversely, improper PAC-learning of C by $\mathcal{H}$ generally becomes easier as $\mathcal{H}$ becomes more expressive: if we can efficiently represent $h \in \mathcal{H}$ by some $h' \in \mathcal{H}'$, then we can always return h' instead of h. We can now sensibly ask the question: *"what are the most expressive representations that are efficiently PAC-learnable?"* The answer can be visualized in the venn diagram illustrating the relationships between the expressive power of the classes of representations we have seen thus far, see Figure 4.2.

Nevertheless, something is lost by this concession: For example, we can solve satisfiability for a three-term DNF in linear time simply by identifying a term that doesn't include a pair x and $\neg x$ and using assignments that satisfy each literal in the term. Satisfiability for 3CNF formulas, by contrast, is NP-complete, and hence presumably intractable. For some applications, a 3CNF representation alone would therefore not suffice whereas we might have been able to work with a three-term DNF efficiently. In general, operations on more expressive representations suffer greater computational complexity. Indeed, we see this clearly in the reductions establishing NP-completeness, where more expressive representations inherit hard instances by translation from any less-expressive representation. So although learning with $\mathcal{H}'$ that is more expressive than $\mathcal{H}$ may be easier, it is generally harder to support subsequent operations on $\mathcal{H}'$ than it would have been with $\mathcal{H}$.

4.7 Cryptographic hardness of improper learning.

Our ability to escape NP-hardness by using more expressive representations raises the tantalizing possibility that perhaps every representable property is PAC-learnable. Unfortunately this is not true, and we will now show that even such "improper" learning

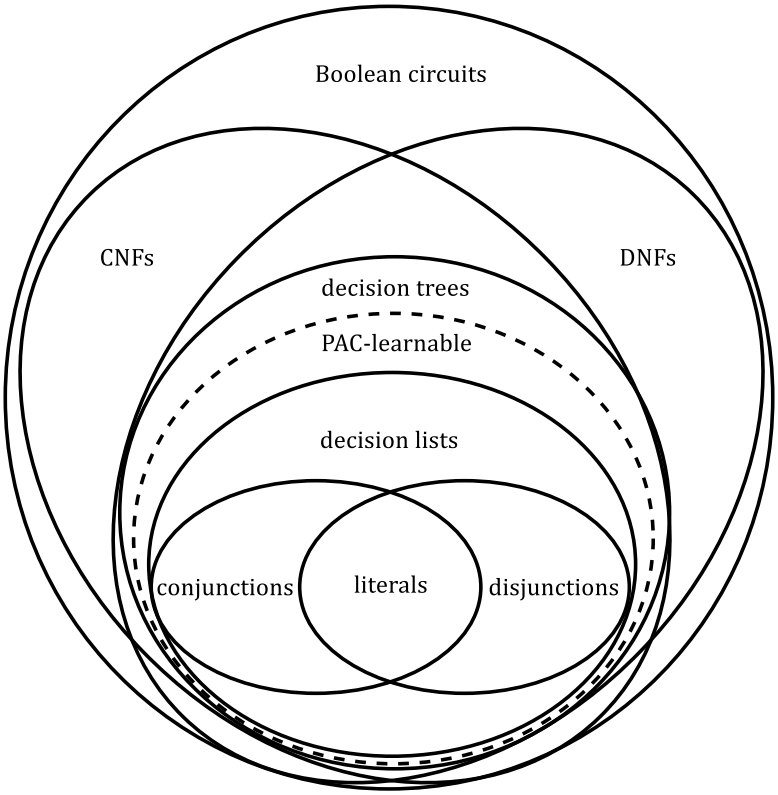


Figure 4.2 Relationships among classes of representations. The efficiently PAC-learnable representations are represented by the dashed line, the precise extent of which remains unknown.

has surprisingly severe limits. These limits are still based on conjectures, as before, but we will here make use of some stronger conjectures than P $\neq$ NP. Instead, we will use the conjectures concerning the hardness of specific problems that provide the foundations of modern cryptography.

Indeed, it is natural that learning and cryptography should be related by opposition to one another: learning is the task of discovering patterns in data, whereas cryptography is the task of obscuring such patterns. In this section, we will formalize the connections and make the relationship precise. This connection turns out to have been fruitful for both disciplines.

A critical functionality in modern cryptography is *public-key cryptography*. This allows one party, "Alice," to initiate secure communication with a second party, "Bob," with whom she has had no prior interaction and in particular shares no secrets. For example, if Bob is selling goods over the Internet, Alice needs to communicate her payment details to him securely, so that an eavesdropper, "Eve," cannot intercept that information and use it to charge payments to Alice's account and so on. Informally, Alice and Bob achieve this as follows: Bob uses a *key-generation algorithm* to generate a *public key* that may be used to send him secure messages. Bob makes

this public key publicly available. Alice, once in possession of the public key, uses an *encryption algorithm* that takes the public key, along with the *message* she wishes to send to Bob, and outputs a *ciphertext*. Alice sends that ciphertext over the Internet to Bob. Upon receiving the ciphertext, Bob uses a *decryption algorithm* that takes the ciphertext along with a second, *private key* that was also produced by the key-generation algorithm, and outputs Alice's message.

The system is *semantically secure* if Eve cannot distinguish between the ciphertexts produced by any two distinct messages, even if she possesses the public key. Note that this means that the system must be randomized, or else Eve could simply encrypt some messages of interest and compare them to the ciphertext sent by Alice. Indeed, the generation of the keys must be randomized. At the same time, the ciphertexts must be distinguishable to Bob, who holds the secret key. Security thus relies on the limits of Eve's computational abilities. The key-generation algorithm is provided a *security parameter* that controls how hard the system is to break; typically this is directly related to the size of the keys.

Definition 4.16 A *public-key encryption scheme* consists of three algorithms:

1. $\text{Gen}(n)$: the *key-generation algorithm* that, given *security parameter* n, outputs a pair: a *secret key* sk and a *public key* pk.
2. $\text{Enc}(pk, m)$: the *encryption algorithm* that, given a public key pk and a *message* m from the scheme's *message space* $\mathcal{M}$, outputs a *ciphertext* c.
3. $\text{Dec}(sk, c)$: the *decryption algorithm* that, given a secret key sk and a ciphertext c, returns a message m'.

The scheme must satisfy that for any pair of keys (sk, pk) produced by $\text{Gen}(n)$ and any ciphertext c produced by $\text{Enc}(pk, m)$ for any message $m \in \mathcal{M}$, $\text{Dec}(sk, c) = m$.

The scheme is *semantically secure* if for any two messages $m_0, m_1 \in \mathcal{M}$ and any polynomial-time probabilistic algorithm A with Boolean output, for every polynomial $p(n)$, for sufficiently large n,

$$\left| \Pr_{\substack{(sk,pk)\leftarrow\text{Gen}(n) \\ c\leftarrow\text{Enc}(pk,m_0)}} [A(1^n, pk, c) = 1] - \Pr_{\substack{(sk,pk)\leftarrow\text{Gen}(n) \\ c\leftarrow\text{Enc}(pk,m_1)}} [A(1^n, pk, c) = 1] \right| < \frac{1}{p(n)}.$$

The connection to learning is that if the decryption algorithm were computing a learnable function, the scheme could not be semantically secure: Eve can use the public key to generate examples by encrypting m_0 and m_1, which she may use to learn to distinguish the respective ciphertexts.

Theorem 4.17 *If C is a class of properties such that for some public-key encryption scheme and some pair of messages $m_0, m_1 \in \mathcal{M}$, for any (pk, sk) produced by $\text{Gen}(n)$, some $d_{sk,m_0,m_1} \in C$ satisfies $d_{sk,m_0,m_1}(\text{Enc}(pk, m_0)) = 0$ and $d_{sk,m_0,m_1}(\text{Enc}(pk, m_1)) = 1$, then C is not improperly PAC-learnable with $\epsilon, \delta \leq 1/8$ in time polynomial in n.*

Proof Suppose for contradiction that a PAC-learning algorithm L for such a class C exists, running in time $t(n)$ for some polynomial t. Then the public-key encryption

Input: Public key pk and challenge ciphertext c such that for $(sk, pk) \leftarrow \mathrm{Gen}(1^n)$, $c \leftarrow \mathrm{Enc}(pk, m^*)$ for $m^* \in \{m_0, m_1\}$.
Output: $b \in \{0,1\}$ such that $m^* = m_b$
begin
  for $i = 1, \ldots, t(n)$ **do**
    $b_i \leftarrow \{0,1\}$ uniformly at random
    $x_i \leftarrow \mathrm{Enc}(pk, m_{b_i})$
  end
  $d \leftarrow L((x_1, b_1), \ldots, (x_{t(n)}, b_{t(n)}))$
  return $d(c)$.
end
Algorithm 3: Adversary using learning to decrypt public-key encryptions

scheme would not be semantically secure: indeed, suppose the algorithm A were as given by Algorithm 3.

Since L is assumed, with probability $1 - 1/8$ to return an efficiently evaluatable hypothesis d that agrees with $\mathrm{Dec}(sk, \cdot)$ with probability $1 - 1/8$ when the input ciphertext is generated by $\mathrm{Enc}(pk, m^*)$ for $m^* \in \{m_0, m_1\}$ chosen uniformly at random, by a union bound over the two $1/8$ failure probabilities,

$$\begin{aligned}
1 - \frac{1}{4} &\leq \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_b)\\ b\in\{0,1\}}}[d(c) = b] \\
&= \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_1)}}[d(c) = 1] \cdot \frac{1}{2} + \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_0)}}[d(c) = 0] \cdot \frac{1}{2} \\
&= \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_1)}}[d(c) = 1] \cdot \frac{1}{2} + \left(1 - \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_0)}}[d(c) = 1]\right) \cdot \frac{1}{2} \\
\Rightarrow \frac{1}{2} &\leq \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_1)}}[d(c) = 1] - \Pr_{\substack{(sk,pk)\leftarrow\mathrm{Gen}(n)\\ c\leftarrow\mathrm{Enc}(pk,m_0)}}[d(c) = 1]
\end{aligned}$$

Thus, since $A(1^n, pk, c)$ returns $d(c)$ in time polynomial in n, the public-key encryption scheme could not have been semantically secure. □

Thus, if secure public-key cryptography is indeed possible, then we know that some representations are not efficiently learnable, even improperly:

Corollary 4.18 *If there exist public-key encryption schemes implementable by polynomial-size Boolean circuits, then Boolean circuits are not efficiently PAC-learnable.*

Furthermore, we see that the more efficiently decryption can be implemented, the weaker the PAC-learnable representations are. As we will see, unfortunately for us, cryptography turns out to be quite efficient.

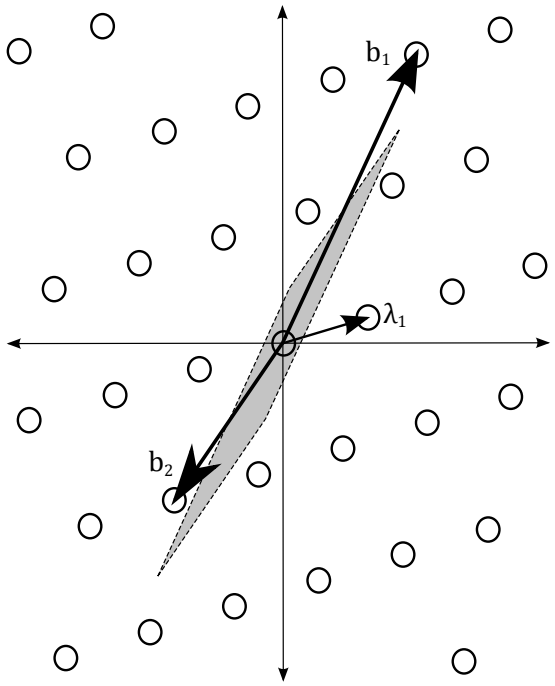


Figure 4.3 Example lattice in 2d given by the basis $\{b_1, b_2\}$. A fundamental parallelipiped (shaded) and the shortest vector λ_1 are shown.

4.8 Learning of intersections of halfspaces is as hard as finding short vectors in lattices.

Modern cryptography provides proofs of security based on assumptions that specific, "famous" computational problems do not have efficient algorithms. One example of such a problem that is used as a foundation for current public-key encryption methods is the (approximate) *shortest vector problem* for integer lattices.

Definition 4.19 An n-dimensional *lattice* $\mathcal{L}$ is a countably infinite subset of $\mathbb{R}^n$ that is closed under addition and subtraction: for $u, v \in \mathcal{L}$, $u \pm v \in \mathcal{L}$. (Notice, therefore, $\vec{0} \in \mathcal{L}$.) A *basis* $B = \{b_1, \ldots, b_d : b_i \in \mathbb{R}^n\}$ for a lattice $\mathcal{L}$ is a linearly independent set of vectors for which $\mathcal{L} = \{\sum_{i=1}^{d} a_i b_i : a_1, \ldots, a_d \in \mathbb{Z}\}$. For a basis B, we let $\mathcal{L}(B)$ denote the corresponding lattice. The *centered fundamental parallelepiped* of B, $\mathcal{P}(B)$, is the set $\{\sum_{i=1}^{d} y_i b_i : y_1, \ldots, y_m \in [-\frac{1}{2}, \frac{1}{2})\}$. Note that for every $\vec{v} \in \mathbb{R}^n$, there exists a unique $\vec{u} \in \mathcal{P}(B)$ and $\vec{w} \in \mathcal{L}(B)$ such that $\vec{v} = \vec{u} + \vec{w}$. We write $\vec{u} = \vec{v} \bmod \mathcal{L}(B)$.

Definition 4.20 The length of the *shortest vector* of a lattice $\mathcal{L}$ is $\lambda_1(\mathcal{L}) = \min_{\vec{v} \in \mathcal{L}} \|\vec{v}\|_2$. The *$\gamma$-approximate shortest vector problem*, GapSVP_γ, is: given a basis B and number d, determine whether $\lambda_1(\mathcal{L}(B)) \leq d$ or $\lambda_1(\mathcal{L}(B)) \geq \gamma \cdot d$. (We do not consider instances where $\lambda_1(\mathcal{L}(B))$ is between d and $\gamma \cdot d$, and an algorithm for GapSVP_γ may produce arbitrary output in these cases.)

The approximate shortest vector problem may appear easy at first glance, but the difficulty that arises is that the same lattice $\mathcal{L}$ may be generated by many bases, which obscure the structure of $\mathcal{L}$. Indeed, the best known polynomial-time algorithms can only achieve a $2^{O(n)}$-approximation, and the problem is NP-hard for constant approximations. For cryptography, we are interested in the regime where γ is polynomially related to the dimension n. Unlike many other problems used in cryptography, the approximate shortest vector problem is *not* known to be solved efficiently by quantum algorithms, and encryption based on it is conjectured to remain secure against adversaries with quantum computers.

Interestingly, some public-key encryption schemes that may be shown to be secure assuming the hardness of GapSVP$_\gamma$ were inspired by the observed difficulty of PAC-learning representations of linear functions in the presence of label noise, as we will see. In the following we will let $n \bmod p$ for integers n and p denote the remainder when dividing n by p, i.e., the integer r in the range $\{0, \ldots, p-1\}$ such that $n = q \cdot p + r$. Note that for any two integers m and n, if $m \bmod p = r_1$, $n \bmod p = r_2$, and $(n+m) \bmod p = r_3$, since $m = q_1 p + r_1$ and $n = q_2 p + r_2$, $m+n = (q_1+q_2)p + (r_1+r_2)$. Since we also had $(m+n) = q_3 p + r_3$, observe that $r_3 = (r_1 + r_2) \bmod p$—or in other words, $(m \bmod p + n \bmod p) \bmod p = (m+n) \bmod p$. We similarly have $[(m \bmod p)(n \bmod p)] \bmod p = (mn) \bmod p$; that is, applying "mod$p$" to the intermediate values of a calculation gives the same result as if we had performed the evaluation using standard integer arithmetic and simply taken the remainder of the final result. The upshot is that this enables arithmetic on finite representations that handles overflow in an associative way, and also enables us to consider a uniform distribution on integers, by simply taking the uniform distribution on the remainders $0, \ldots, p-1$.

Definition 4.21 In the *Learning With Errors* problem parameterized by integers p, m, n, and σ, vectors $\vec{x}^{(1)} \ldots, \vec{x}^{(m)}$ are drawn uniformly at random from $\{0, \ldots, p-1\}^n$ and a *secret* vector $\vec{s}$ is drawn uniformly at random from $\{0, \ldots, p-1\}^n$. For $\vec{x}^{(i)}$, we put the integer label $b^{(i)} = \langle \vec{s}, \vec{x}^{(i)} \rangle + e^{(i)} \bmod p$ where the *error* e_i is sampled independently from a mean-0 variance-σ^2 Gaussian conditioned on taking an integer value, i.e., the integer z has density proportional to $e^{-z^2/2\sigma^2}$ – we refer to this as the *discrete Gaussian* distribution. Then, given $(\vec{x}^{(1)}, b^{(1)}), \ldots, (\vec{x}^{(m)}, b^{(m)})$ the task is to find $\vec{s}$.

As we'll sketch briefly, algorithms for solving the Learning With Errors problem yield algorithms for GapSVP, so if GapSVP is intractable, so is Learning With Errors for appropriate choices of parameters. In turn, Learning With Errors serves as the basis for the following public-key encryption scheme:

Definition 4.22 *Regev's lattice-based encryption* is the following public-key encryption scheme with parameters m, n, p and $\gamma(n)$:

1. The private key is $\vec{s}$ chosen uniformly at random from $\{0, \ldots, p-1\}^n$. The public key consists of m pairs $(\vec{a}_i, b_i)$ where $\vec{a}_i$ is chosen uniformly at random from $\{0, \ldots, p-1\}^n$ and $b_i = \langle \vec{s}, \vec{a}_i \rangle + e_i \bmod p$ for e_i sampled from the discrete Gaussian distribution with standard deviation $\sigma = 1/\gamma(n)$.

2. Encryption is performed on a single-bit message by choosing $S \subseteq \{1, \ldots, m\}$ uniformly at random. To encrypt 0, output $(\sum_{i \in S} \vec{a}_i \bmod p, \sum_{i \in S} b_i \bmod p)$. To encrypt 1, output $(\sum_{i \in S} \vec{a}_i \bmod p, \lfloor p/2 \rfloor + \sum_{i \in S} b_i \bmod p)$.
3. Decryption of the ciphertext $(\vec{a}, b)$ outputs 1 if $b - \langle \vec{a}, \vec{s} \rangle \bmod p$ is in the range $[p/4, 3p/4]$. It outputs 0 otherwise.

Overall, Regev's lattice-based encryption is secure if GapSVP is intractable:

Theorem 4.23 *For p prime in the range $(n^2, 2n^2)$, $m = 5(n+1)\log_2 p$, and $\gamma(n)$ a function that is $\omega(\sqrt{n}\log n)$, if Regev's lattice-based encryption is not semantically secure, then there is a polynomial-time algorithm for GapSVP with approximation $\tilde{O}(n\gamma(n))$ (where $\tilde{O}$ omits $\log n$ factors). In particular, for $\gamma(n) = \sqrt{n}\log_2^2 n$, the probability of a decryption error in Regev's encryption scheme is $2^{-\Omega(\log^3 n)}$, and would yield a polynomial-time algorithm for $\text{GapSVP}_{\tilde{O}(n^{3/2})}$.*

At the same time, the decryption of ciphertexts in Regev's scheme can be computed by extremely simple circuits.

Definition 4.24 *Halfspaces* (also known as *linear threshold functions*) are the family of functions $h : \mathbb{R}^n \to \{0, 1\}$ parameterized by $w_1 \ldots, w_n \in \mathbb{R}$ and $t \in \mathbb{R}$, given by $h(\vec{x}) = [\langle \vec{w}, \vec{x} \rangle \geq t]$. Degree-$d$ *polynomial threshold functions* are functions parameterized by a degree-d n-variate polynomial $p(\vec{x})$ and $t \in \mathbb{R}$, analogously given by $[p(\vec{x}) \geq t]$. *Intersections of halfspaces* are functions given by a conjunction (AND) of halfspaces, and analogously, *intersections of degree-d polynomial threshold functions* are conjunctions of degree-d polynomial threshold functions.

This is the first time we've used an instance space beyond Boolean attributes, so a few remarks are in order. First, we observe that if we interpret the Boolean values 0 and 1 as the corresponding real numbers, a conjunction $x_1 \wedge \cdots \wedge x_k$ of Boolean values is identical to the function computed by the corresponding monomial $x_1 \cdots x_k$. So, analogously to the reduction in Theorem 4.14, for any fixed integer d, learning degree-d polynomial threshold functions reduces to learning halfspaces, and learning intersections of degree-d polynomial threshold functions reduces to learning intersections of halfspaces. Although we've written the functions as parameterized by real numbers and taking real numbers as inputs (as this lends itself most naturally to the geometric interpretation we're considering here), in practice we'll restrict attention to instances where the inputs or parameters are rational or integer valued. We discuss these technicalities and the consequences for learnability in more detail in Appendix A.

We finally arrive at the main result of this section: we will show that decryption can be computed by intersections of degree-2 polynomial threshold functions; in turn, this yields that if the shortest vector problem is intractable, then there is no polynomial-time PAC-learning algorithm for intersections of halfspaces:

Theorem 4.25 *If Regev's lattice-based encryption is semantically secure, then intersections of halfspaces are not efficiently PAC-learnable; in particular, therefore if there is no polynomial-time algorithm for $\text{GapSVP}_{\tilde{O}(n^{3/2})}$, then intersections of halfspaces are not efficiently PAC-learnable.*

Proof In Regev's encryption scheme, $(\vec{a}, b)$ decrypts to 1 if $b - \langle \vec{a}, \vec{s} \rangle \bmod p$ is in the range $[p/4, 3p/4]$. We will rewrite this expression using the binary expansions of $\vec{a}$ and b which, recall, take values in $\{0, \ldots, p-1\}$: Let $a_i = \sum_{j=0}^{\log_2 p} 2^j x_{i,j}$ for $x_{i,j} \in \{0,1\}$ and $b = \sum_{j=0}^{\log_2 p} 2^j y_j$ for $y_j \in \{0,1\}$. Now, we can write the expression $b - \langle \vec{a}, \vec{s} \rangle \bmod p$ as $\sum_{j=0}^{\log_2 p} 2^j y_j - \sum_{i=1}^{n} s_i \sum_{j=0}^{\log_2 p} 2^j x_{i,j} \bmod p$. We now use the properties of modular arithmetic to replace each 2^j by $2^j \bmod p$ and $-s_i 2^j$ by $-s_i 2^j \bmod p$, which now lie in the range $\{0, \ldots, p-1\}$. In summary, we obtain $\sum_{j=0}^{\log_2 p} (2^j \bmod p) y_j + \sum_{i=1}^{n} \sum_{j=0}^{\log_2 p} (-s_i 2^j \bmod p) x_{i,j}$, which we denote by z. z takes a value in the range $\{0, \ldots, (p-1)(n+1)\log_2 p\}$.

Recalling the definition of mod$p$, we observe that $(\vec{a}, b)$ decrypts to 0 if, for $z = q \cdot p + r$ for $r \in \{0, \ldots, p-1\}$, $r \notin [\frac{p}{4}, p - \frac{p}{4}]$, which is the same as $r \in [0, \frac{p-1}{4}] \cup [p - \frac{p-1}{4}, p-1]$ since r is an integer; in turn, since $r = z - q \cdot p$, this is equivalent to z falling in $[q \cdot p - \frac{p-1}{4}, q \cdot p + \frac{p-1}{4}]$ for some integer q.

The key observation now is that intervals can be represented by a degree-2 polynomial threshold function: indeed, the interval $[u, v]$ can be represented by $(z - \frac{u+v}{2})^2 \leq \left(\frac{v-u}{2}\right)^2$. Similarly, for integer z, u, and v, z not falling in $[u, v]$ can be respresented by $(2z - (u+v))^2 \geq (v-u)^2 + 1$ since the expressions all take integer values.

In summary, we obtain a decryption of 1 if we do not obtain a 0, which occurs if z does not fall in $[q \cdot p - \frac{p-1}{4}, q \cdot p + \frac{p-1}{4}]$ for any integer q: that is, if

$$\left(\sum_{j=0}^{\log_2 p} 2(2^j \bmod p) y_j + \sum_{i=1}^{n} \sum_{j=0}^{\log_2 p} 2(-s_i 2^j \bmod p) x_{i,j} - 2q \cdot p \right)^2 \geq \left(\frac{p-1}{2} \right)^2 + 1$$

for all q. But, since z takes a value in the range $\{0, \ldots, (p-1)(n+1)\log_2 p\}$, we only need to consider q such that qp lies in the range $[0, (p-1)(n+1)\log_2 p]$, i.e., q from the set $\{0, \ldots, (n+1)\log_2 p\}$. We thus can compute the decryption by an intersection of $1 + (n+1)\log_2 p$ degree-2 polynomial threshold functions when $(\vec{a}, b)$ are represented in binary. Therefore, by Theorem 4.17, if Regev's lattice-based encryption scheme is semantically secure, intersections of degree-2 polynomial threshold functions are not efficiently learnable. Since intersections of degree-2 polynomial threshold functions may be reduced to learning intersections of halfspaces, intersections of halfspaces are not efficiently learnable either. Finally, since Theorem 4.23 establishes that Regev's lattice-based encryption scheme is secure if there is no polynomial-time algorithm for GapSVP$_{\tilde{O}(n^{3/2})}$, we obtain that intersections of halfspaces are also not efficiently learnable under the same hypothesis. □

Before continuing, we note that an intersection of halfspaces may be represented as a kind of two-layer neural network with 0-1 "activation".

Definition 4.26 A *neural network* with *activation* function $\alpha(x, t)$ is a circuit in which each gate computes a function of the form $g(\vec{x}) = \alpha(\langle \vec{w}, \vec{x} \rangle, t)$ for some real-valued *weight* vector $\vec{W}$ and *threshold* t. Standard choices for activation are 0-1, $[x \geq t]$, Sigmoid $\frac{1}{1+e^{-(x-t)/\sigma}}$ (for some $\sigma \in \mathbb{R}$), and *Rectified Linear Unit (ReLU)*, $\max\{0, x-t\}$.

Indeed, note that the AND of $x_1, \ldots, x_n$ may be represented as $[\sum_{i=1}^{n} x_i \geq n]$. We thus immediately obtain:

Corollary 4.27 *If there is no polynomial-time algorithm for* $\textsc{GapSVP}_{\tilde{O}(n^{3/2})}$*, then two-layer neural networks with 0-1 activation are not efficiently learnable.*

Since the standard methods for learning neural networks rely on computing gradients, it is natural to suspect that the difficulty with learning neural networks with 0-1 activation is due to the lack of useful gradients, which are 0 almost everywhere, except at the discontinuity where the function jumps from 0 to 1. It turns out that this is not the problem, and all of the standard activations give representations that cannot be learned efficiently.

Corollary 4.28 *If there is no polynomial-time algorithm for* $\textsc{GapSVP}_{\tilde{O}(n^{3/2})}$*, then two-layer neural networks with ReLU activation are not efficiently learnable.*

Proof Observe that the distribution on inputs in Theorem 4.25 is integer-valued (actually $\{0,1\}$-valued), as are the weights. For all integers z and t, the $\{0,1\}$-valued function $[z \geq t]$ is identical to $\max\{0, z-t\} - \max\{0, z-t-1\}$. Likewise, an AND of Booleans $x_1, \ldots, x_k$ may be represented as $\max\{0, \sum_{i=1}^{k} x_i - (k-1)\}$. Thus, the intersection of k halfspaces

$$[\langle \vec{w}^{(1)}, \vec{x} \rangle \geq t^{(1)}] \cap \cdots \cap [\langle \vec{w}^{(k)}, \vec{x} \rangle \geq t^{(k)}]$$

may be represented as the following two-layer neural network with ReLU activation:

$$\max\left\{0, \sum_{i=1}^{k} [\max\{0, \langle \vec{w}^{(i)}, \vec{x} \rangle - t^{(i)}\} - \max\{0, \langle \vec{w}^{(i)}, \vec{x} \rangle - t^{(i)} - 1\}] - (k-1)\right\}$$

Thus, if intersections of halfspaces are not efficiently learnable, two-layer neural networks with ReLU activations are also not efficiently learnable. □

We leave the argument for sigmoid activation as an exercise.

The intractability of learning neural networks, captured by Corollaries 4.27 and 4.28, stands in stark contrast to the empirical success of neural network methods in practice. We can reconcile these seemingly contradictory facts by recalling that in improper learning (cf. Definition 4.15), we are learning properties representable by some class of representations $\mathcal{C}$ using a possibly more expressive hypothesis representation $\mathcal{H}$. Remember, Theorem 4.14 shows that we can learn 3-term DNF using 3CNF representations, even though learning using 3-term DNF representations is NP-hard. Here, we have shown that it is not possible to learn *all* properties representable by two-layer neural networks with either 0-1 or ReLU activation (i.e., taking $\mathcal{C}$ to be one of these families) using *any* efficiently evaluatable hypothesis representation $\mathcal{H}$, since these neural networks can represent the decryption function for Regev's lattice-based encryption scheme. Neural network learning methods, by contrast, use e.g., ReLU or sigmoid neural networks as the hypothesis representation $\mathcal{H}$ to learn some other, *necessarily less expressive* class of representations $\mathcal{C}$. These methods successfully

produce neural network representations as output, but do not learn arbitrary properties representable by neural networks.

To reiterate: in general, by making the class of hypothesis representations $\mathcal{H}$ more expressive, we obtain an easier learning problem; again, Theorem 4.14 demonstrates such an example, where we obtained a tractable learning problem by using a hypothesis representation $\mathcal{H}$ that is strictly more expressive than the family of properties C to be learned. The value of neural networks is that they serve as a particularly expressive hypothesis class for which particularly effective algorithms have been developed. Yet, not all functions that can be represented by neural networks can be learned by these methods, assuming security of the lattice-based encryption schemes. Furthermore, the expressive power of these representations poses a hinderance to their use in subsequent tasks, such as the reasoning problems we'll consider in the next chapter.

4.8.1 Security from the shortest vector problem.

We will conclude this section by sketching the sequence of reductions that establish the security of Regev's lattice-based encryption given that GapSVP is intractable, essentially describing the proof of Theorem 4.23. Unfortunately, the full details of these reductions is beyond the scope of this book and we must refer the interested reader to the original articles.

We begin by connecting the shortest vector problem to the Learning With Errors problem via a third problem, *"Bounded Distance Decoding"*:

Definition 4.29 *α-Bounded Distance Decoding*, denoted BDD_α, is the following search problem. We are given a basis B for an integer lattice $\mathcal{L}(B)$, and a vector $\vec{t}$ obtained as the sum of $\vec{v} \in \mathcal{L}(B)$ and an error $\vec{e}$ of norm $\|\vec{e}\|_2 \leq \alpha\lambda_1(\mathcal{L}(B))$, i.e., a α fraction of the length of the shortest vector. The task is to return $\vec{v}$.

4.8.1.1 From shortest vectors to Bounded Distance Decoding.

GapSVP_γ reduces to $\mathrm{BDD}_{\frac{1}{\gamma}\sqrt{\frac{n}{\log_2 n}}}$ for $\gamma > 2\sqrt{n/\log_2 n}$ easily: given an instance of GapSVP_γ (B, d), we sample $m = n/\delta$ vectors $\vec{s}^{(1)}, \ldots, \vec{s}^{(m)}$ uniformly at random from vectors of length $d\sqrt{n/\log_2 n}$, put $\vec{t}^{(i)} = \vec{s}^{(i)} \bmod \mathcal{L}(B)$, and use $(B, \vec{t}^{(i)})$ as the input to $\mathrm{BDD}_{\frac{1}{\gamma}\sqrt{\frac{n}{\log_2 n}}}$ to obtain $\vec{v}^{(i)} \in \mathcal{L}(B)$. We report $\lambda_1(\mathcal{L}(B)) \leq d$ if we obtain $\vec{s}^{(i)}$ from each run and otherwise report $\lambda_1(\mathcal{L}(B)) \geq \gamma d$.

Indeed, if $\lambda_1(\mathcal{L}(B)) \geq \gamma d$, since $d\sqrt{\frac{n}{\log_2 n}} \leq \frac{\lambda_1(\mathcal{L}(B))}{\gamma}\sqrt{\frac{n}{\log_2 n}}$ and $\frac{1}{\gamma}\sqrt{\frac{n}{\log_2 n}} < \frac{1}{2}$, $(B, \vec{t}^{(i)})$ is a valid instance of $\mathrm{BDD}_{\frac{1}{\gamma}\sqrt{\frac{n}{\log_2 n}}}$ and furthermore, $\vec{s}^{(i)}$ is the unique vector within radius $\lambda_1(\mathcal{L}(B))/2$ since the minimum distance between vectors is $\lambda_1(\mathcal{L}(B))$. Thus, in this case, all runs must return the respective $\vec{s}^{(i)}$.

By contrast, if $\lambda_1(\mathcal{L}(B)) \leq d$, there is at least one other lattice vector $\vec{r}^{(i)}$ such that $\|\vec{s}^{(i)} - \vec{r}^{(i)}\| = \lambda_1(\mathcal{L}(B))$. In this case, whenever $\vec{t}^{(i)}$ is within distance $d\sqrt{n/\log_2 n}$ of $\vec{r}^{(i)}$, $\vec{t}^{(i)}$ is equally likely to be obtained from $\vec{s}^{(i)}$ or $\vec{r}^{(i)}$, so the algorithm for $\mathrm{BDD}_{\frac{1}{\gamma}\sqrt{\frac{n}{\log_2 n}}}$ can only return our $\vec{s}^{(i)}$ ("guess right") with probability at most $1/2$.

Indeed, for such large γ, we obtain these points with high probability, and thus the bounded distance decoding algorithm succeeds at guessing every $\vec{s}^{(i)}$ with negligible probability. In particular, now, $\text{GapSVP}_{\tilde{O}(n^{3/2})}$ reduces to $\text{BDD}_{\tilde{O}(1/n)}$.

4.8.1.2 From Bounded Distance Decoding to Learning With Errors.

We now show how to reduce the Bounded Distance Decoding problem to an instance of Learning With Errors. The main issue we must deal with is that Bounded Distance Decoding is a worst-case problem with potentially large inputs, whereas Learning With Errors features instances uniformly distributed over $\{0, \ldots, p-1\}$ perturbed by Gaussian noise. The key to the reduction is that if the variance of a Gaussian is large enough, then when we examine the remainders (reduce modulo p), the result is nearly uniformly distributed over $\{0, \ldots, p-1\}$: variance $4p^2 \ln(2(1+\frac{1}{\epsilon})n)$ is adequate to ensure the distribution of an n-variate Gaussian is 4ϵ-close to uniform in statistical distance. In particular, in the output of Algorithm 4, $\vec{x}^{(1)}, \ldots, \vec{x}^{(m)}$ are 4ϵ-close to independent, uniform random vectors from $\{0, \ldots, p-1\}^n$.

Input: Lattice basis B, $\vec{t} = B\vec{s} + \vec{e}$, parameter ϵ. (Assume $\|\vec{e}\|_2$ is also given.)
Output: Learning With Errors instance $(\vec{x}^{(1)}, b^{(1)}), \ldots, (\vec{x}^{(m)}, b^{(m)})$ i.e., where each $\vec{x}^{(i)}$ is close to an independent uniform random vector $\{0, \ldots, p-1\}^n$, and $b^{(i)} = \langle \vec{s}, \vec{x}^{(i)} \rangle + e'^{(i)} \bmod p$ with e' close to a discrete Gaussian on $\{0, \ldots, p-1\}$.

begin
  for $i = 1, \ldots, m$ **do**
    $\vec{x}^{(i)} \leftarrow \lfloor \vec{g} \rfloor \bmod p$ where each g_j is sampled independently from a mean-0 Gaussian with $\sigma^2 \geq 4p^2 \ln(2(1+\frac{1}{\epsilon})n)$
    $b^{(i)} \leftarrow (B^{-1}\vec{t})^\top \vec{x}^{(i)} + e'^{(i)} \bmod p$ where $e'^{(i)}$ is sampled from a mean-0 Gaussian with $\sigma'^2 = 2\|e\|_2^2 \|B^{-1}\|_2^2 \ln(2(1+\frac{1}{\epsilon})n)$
  end
  return $(\vec{x}^{(1)}, b^{(1)}), \ldots, (\vec{x}^{(m)}, b^{(m)})$.
end

Algorithm 4: Reducing Bounded Distance Decoding to Learning With Errors

At the same time, the sum of two Gaussians is itself a Gaussian with variance equal to the sum of the two variances; more generally, if g_1 is a mean-0 Gaussian with variance σ_1^2 and g_2 is a mean-0 Gaussian with variance σ_2^2, then $g_3 = \alpha g_1 + \beta g_2$ is a mean-0 Gaussian with variance $\alpha^2\sigma_1^2 + \beta^2\sigma_2^2$. This will allow us to ensure the error has a suitable distribution: Indeed, observe that $b^{(i)} = (B^{-1}\vec{t})^\top \vec{x}^{(i)} + e' \bmod p$ can be rewritten as follows. First, since $\vec{t} = B\vec{s} + \vec{e}$, $B^{-1}\vec{t} = \vec{s} + B^{-1}\vec{e}$ and $b^{(i)} = \langle \vec{s}, \vec{x}^{(i)} \rangle + \langle B^{-1}\vec{e}, \vec{x}^{(i)} \rangle + e' \bmod p$. Now, we note that e' may be written as $\langle B^{-1}\vec{e}, \vec{g}' \rangle$ where $\vec{g}'$ is a vector of n independent mean-0 Gaussians with variance $\frac{\sigma'^2}{\|B^{-1}\vec{e}\|_2^2}$: indeed, this linear combination is distributed identically to e', mean-0 with variance σ'^2. Now, observe that σ' has been chosen so that $\frac{\sigma'^2}{\|B^{-1}\vec{e}\|_2^2} \geq 2\ln(2(1+\frac{1}{\epsilon})n)$, where $b^{(i)}$ may now be written as $\langle \vec{s}, \vec{x}^{(i)} \rangle + \langle B^{-1}\vec{e}, \vec{x}^{(i)} + \frac{1}{\|B^{-1}\vec{e}\|_2^2}\vec{g}' \rangle$. Since both $\vec{x}^{(i)}$ and

$\frac{1}{\|B^{-1}\vec{e}\|_2^2}\vec{g}'$ have variance at least $2\ln(2(1+\frac{1}{\epsilon})n)$, it may be shown that the sum of the rounded Gaussian $\vec{x}^{(i)}$ and continuous Gaussian $\frac{1}{\|B^{-1}\vec{e}\|_2^2}\vec{g}'$ are 3ϵ-close to a vector of independent mean-0 (continuous-valued) Gaussians with the sum of the variances. Thus, finally, $b^{(i)} = \langle \vec{s}, \vec{x}^{(i)} \rangle + e^{(i)} \bmod p$ where $e^{(i)}$ is a mean-0 Gaussian with variance $\|B^{-1}\vec{e}\|_2^2\sigma^2 + \sigma'^2$. Since the variances all depend only logarithmically on ϵ, we can take ϵ to be a negligible function of n such as $\frac{1}{n^{\log_2 n}}$.

Supposing that $\vec{t}$ has been reduced modulo $\mathcal{L}(B)$ (as in our reduction from GapSVP), $\vec{t}$ lies in the fundamental parallelepiped $\mathcal{P}(B)$. Recall that the lattice points reduce to the vertices of the fundamental parallelepiped, which correspond to $\sum_{i=1}^n \pm\frac{1}{2}B_i$ for various signings of the B_i. (Otherwise we would not have a unique representation.) Since $B\vec{s}$ is a lattice point, $B\vec{s} \bmod \mathcal{P}(B)$ may be written $\sum_{i=1}^n \frac{\Sigma_{i,i}}{2}B_i$ for any signing of the basis vectors Σ (a diagonal ±1 matrix). Therefore, $\sum_{i=1}^n \frac{\Sigma_{i,i}}{2}B_i = -\vec{e} + \vec{t}$. Supposing we are reducing from $\mathrm{BDD}_{\tilde{O}(1/n)}$, $\|\vec{e}\|_2 \leq \tilde{O}(\lambda_1(\mathcal{L}(B))/n)$, so for $B\vec{s} \bmod \mathcal{P}(B)$ (which is the closest vertex) for the corresponding signing Σ, in particular, since $\vec{t}$ lies closest to this vertex in the fundamental parallelepiped, we can write $\vec{t} = \sum_{i=1}^n \Sigma_{i,i}\tau_i B_i$ with $\frac{1}{2} \geq \tau_i \geq 0$. Now, $-B^{-1}\vec{e} + B^{-1}\vec{t} = \sum_{i=1}^n \frac{\Sigma_{i,i}}{2}B^{-1}B_i = \Sigma\frac{\vec{1}}{2}$. At the same time, $B^{-1}\vec{t} = \Sigma\vec{\tau}$. Thus, $-B^{-1}\vec{e} = \Sigma(\frac{1}{2} - \tau_1, \ldots, \frac{1}{2} - \tau_n)$. Moreover, by the bound on $\|\vec{e}\|_2$, each $\tau_i \geq \frac{1}{2} - \frac{1}{n}$. Thus, $\|B^{-1}\vec{e}\|_2 \leq \tilde{O}(1/\sqrt{n})$. For sufficiently large p, $\|B^{-1}\vec{e}\|_2\sigma$ dominates σ' and then $e^{(i)}$ has magnitude $\tilde{O}(p/\sqrt{n})$ with high probability.

4.8.1.3 From Learning With Errors to distinguishing structured from random.

It is helpful now to reduce the Learning With Errors problem to a simpler problem of distinguishing the structured Learning With Errors distribution from simply uniform random labeled instances; precisely, since the $\vec{x}^{(i)}$ in Learning With Errors are already drawn independently and uniformly at random from $\{0, \ldots, p-1\}^n$, we are distinguishing from the Learning With Errors distribution where the labels are noisy evaluations of $\langle \vec{s}, \vec{x}^{(i)} \rangle \bmod p$ from labels that are independently drawn uniformly at random from $\{0, \ldots, p-1\}$.

Definition 4.30 We say that an algorithm that produces Boolean output *succeeds at distinguishing* two distributions if the probability it outputs 1 when given a sample from the first distribution and the probability it outputs 1 on a sample from the second distribution differ by some polynomially bounded amount in terms of the size parameter n.

In particular, we will reduce from the search problem where p is large. Here, we take p to be a power of two for simplicity, but other values of p may be used, following a similar approach. Consider the sequence of distributions D_i for $i = 0, \ldots, k$ where D_i is sampled as follows: we take a sample from the Learning With Errors distribution, sample $r \in \{0, \ldots, p-1\}$ uniformly at random, and add $r \cdot 2^i$ to the label, reducing modulo p. Thus, D_0 has a uniformly random label and D_k is simply the Learning With Errors distribution. If these distributions are distinguished with some probability

$1/d(n)$ by an algorithm Alg, then observe that by the triangle inequality,

$$\frac{1}{d(n)} \leq |\Pr_{x\in D_0}[Alg(x)] - \Pr_{x\in D_k}[Alg(x)]| \leq \sum_{i=1}^{k} |\Pr_{x\in D_{i-1}}[Alg(x)] - \Pr_{x\in D_i}[Alg(x)]|$$

so some pair $i^* - 1$ and i^* must satisfy $|\Pr_{x\in D_{i-1}}[Alg(x)] - \Pr_{x\in D_i}[Alg(x)]| \geq 1/k \cdot d(n)$.

Now, we iteratively find the first i^* bits of the components of $\vec{s}$ as follows: supposing we have found the first $j-1$ bits $\vec{s}_{1:j-1}$, we subtract them off by computing $b \mapsto b - \langle \vec{s}_{1:j-1}, \vec{x}\rangle \bmod p$; note that since $b = \langle \vec{s}, \vec{x}\rangle + e \bmod p$,

$$b - \langle \vec{s}_{1:j-1}, \vec{x}\rangle \bmod p = \langle \vec{s} - \vec{s}_{1:j-1}, \vec{x}\rangle + e \bmod p$$

where the $j-1$ least significant bits of $\vec{s} - \vec{s}_{1:j-1}$ (which we'll denote $\vec{s}_{j:\ell}$) are 0. Now, putting $\vec{u}_i = (0, \ldots, 0, \overbrace{1}^{i}, 0, \ldots, 0)$, we estimate the probability the distinguisher outputs 1 on the input $(\vec{x} + r2^{i^*-j-1}\vec{u}_i \bmod p, b)$ for $(\vec{x}, b)$ drawn from the Learning with Errors distribution and r uniformly random. Observe, if the jth bit of the ith coordinate of $\vec{s}_{j:\ell}$ is 0, then since this ith coordinate may be written $s' \cdot 2^{j+1}$ for $s' \neq 0$, $r' = rs' \bmod p$ is a uniform random element of $\{0, \ldots, p-1\}$, and hence

$$\langle \vec{s}_{j:\ell}, \vec{x} + r2^{i^*-j-1}\vec{u}_i\rangle + e = \langle \vec{s}_{j:\ell}, \vec{x}\rangle + s'2^{i^*-j-1}r2^{j+1} + e = b + r'2^{i^*} \bmod p$$

whereas if the jth bit is 1, the ith coordinate of $\vec{s}_{j:\ell}$ is $s' \cdot 2^j$ for $s' \neq 0$, so we obtain

$$\langle \vec{s}_{j:\ell}, \vec{x} + r2^{i^*-j-1}\vec{u}_i\rangle + e = \langle \vec{s}_{j:\ell}, \vec{x}\rangle + s'2^{i^*-j-1}r2^{j} + e = b + r'2^{i^*-1} \bmod p$$

i.e., respectively, samples from D_{i^*} and D_{i^*-1}. We thus can infer this jth bit.

We note that by adding additional Gaussian noise to the examples drawn from the distribution, we can ensure that i^* is not too small: if the means of two Gaussians with standard deviation σ differ by $\epsilon\sigma$, then they can be distinguished with probability at most $O(\epsilon)$. In particular, if we cannot directly recover all of the bits of $\vec{s}$, then we can add nonnegligible noise, up to standard deviation $\sim p/\sqrt{n}$ and remain an instance of Learning with Errors with $\tilde{O}(p/\sqrt{n})$-bounded noise. Then, for the highest-order bits, we have that every coordinate of $\vec{s}_{i^*:\ell}$ is a multiple of 2^{i^*}; since the standard deviation of the noise of the original distribution is smaller than 2^{i^*} by $\tilde{\Omega}(p/\sqrt{n})$, we can obtain that with high probability simply rounding samples to the nearest multiple of 2^{i^*} will give noiseless examples $(\vec{x}, \langle \vec{s}_{i^*:\ell}, \vec{x}\rangle \bmod p)$. We can then solve directly for $\vec{s}_{i^*:\ell}$ using Gaussian elimination on the resulting system of linear equations.

4.8.1.4 Breaking encryption means distinguishing structured from random.

Finally, we suppose that there is an adversary Adv that violates semantic security of Regev's lattice-based encryption scheme, meaning it obtains a polynomial advantage $1/d(n)$ at distinguishing the encryption of 0 from the encryption of 1, given access to the public key. We consider what happens when, instead of an encryption of 0 $(\sum_{i\in S}\vec{a}_i \bmod p, \sum_{i\in S} b_i \bmod p)$, or an encryption of 1, $(\sum_{i\in S}\vec{a}_i \bmod p, \lfloor p/2\rfloor + \sum_{i\in S} b_i \bmod p)$ (for uniform random subsets S of the m pairs), it is given $(\vec{x}, r)$

for a uniform random $\vec{x}$ and r. By the triangle inequality, it must obtain advantage at least $1/2d(n)$ at distinguishing one of the encryptions from this uniform random distribution. If it is for encryptions of 1, observe that adding $\lceil p/2 \rceil$ modulo p to the second component transforms an encryption of 1 into an encryption of 0 and transforms a uniform random label into another label that still is distributed uniformly over $\{0, \ldots, p-1\}$. Thus, without loss of generality, we suppose that Adv distinguishes encryptions of 0 from uniform random pairs.

We can now use Adv to distinguish structured from random as follows: we take a sample $(\vec{x}^{(1)}, b^{(1)}), \ldots, (\vec{x}^{(m)}, b^{(m)})$ of m examples from our unknown distribution, choose uniform random subsets S, and run Adv with the sample as the public key and $(\sum_{i \in S} \vec{x}^{(i)} \bmod p, \sum_{i \in S} b^{(i)} \bmod p)$ as the "ciphertext" to be decrypted. If the sample is drawn from the Learning With Errors distribution, then this is precisely the distribution of an encryption of 0 with the corresponding public key. On the other hand, if the sample is from the uniform random distribution, then we can show that the label $\sum_{i \in S} b^{(i)} \bmod p$ is exponentially close to uniformly distributed. The gap between the probability Adv outputs 1 on the two distributions is then still polynomially large, and hence Adv succeeds at distinguishing the structured distribution from random.

It thus remains only to argue that for the random distribution, $(\sum_{i \in S} \vec{x}^{(i)} \bmod p, \sum_{i \in S} b^{(i)} \bmod p)$ is exponentially close to uniformly distributed. Here, we will rely on the choice of $m \geq 5(n+1)\log_2 p$.

We observe that the optimal (computationally-unbounded) distinguishing function obtains advantage

$$\begin{aligned}
&\max_{A \subseteq \{0,\ldots,p-1\}^{n+1}} \left| \Pr_S \left[A(\sum_{i \in S} \vec{x}^{(i)} \bmod p, \sum_{i \in S} b^{(i)} \bmod p) = 1 \right] - \Pr_{(\vec{x},r)} [A(\vec{x}, r) = 1] \right| \\
&= \frac{1}{2} \sum_{(\vec{x},r) \in \{0,\ldots,p-1\}^{n+1}} \left| \Pr_S \left[(\sum_{i \in S} \vec{x}^{(i)} \bmod p, \sum_{i \in S} b^{(i)} \bmod p) = (\vec{x}, r) \right] - \frac{1}{p^{n+1}} \right| \\
&\leq \frac{\sqrt{p^{n+1}}}{2} \sqrt{\sum_{(\vec{x},r) \in \{0,\ldots,p-1\}^{n+1}} \left(\Pr_S \left[(\sum_{i \in S} \vec{x}^{(i)} \bmod p, \sum_{i \in S} b^{(i)} \bmod p) = (\vec{x}, r) \right] - \frac{1}{p^{n+1}} \right)^2}
\end{aligned}$$

We can bound this expression as follows. We first consider its expected value over the uniform-random draw of $(\vec{x}^{(1)}, b^{(1)}), \ldots, (\vec{x}^{(m)}, b^{(m)})$. Expanding the square in the last line of the previous calculation and invoking linearity of expectation, we can

separately consider

$$\mathop{\mathbb{E}}_{\{\vec{x}^{(i)},b^{(i)}\}}\left[\sum_{\substack{(\vec{x},r)\in\\ \{0,\ldots,p-1\}^{n+1}}}\Pr_S\left[(\sum_{i\in S}\vec{x}^{(i)} \bmod p, \sum_{i\in S} b^{(i)} \bmod p) = (\vec{x},r)\right]^2\right],$$

$$\mathop{\mathbb{E}}_{\{\vec{x}^{(i)},b^{(i)}\}}\left[\Pr_{(\vec{x}',r')}[(\vec{x}',r') = (\vec{x},r)] \cdot \sum_{\substack{(\vec{x},r)\in\\ \{0,\ldots,p-1\}^{n+1}}}\Pr_S\left[(\sum_{i\in S}\vec{x}^{(i)} \bmod p, \sum_{i\in S} b^{(i)} \bmod p) = (\vec{x},r)\right]\right],$$

$$\text{and } \mathop{\mathbb{E}}_{\{\vec{x}^{(i)},b^{(i)}\}}\left[\sum_{\substack{(\vec{x},r)\in\\ \{0,\ldots,p-1\}^{n+1}}}\Pr_{(\vec{x}',r')}[(\vec{x}',r') = (\vec{x},r)]^2\right]$$

The first expression gives the probability that we obtain the same value of $(\sum_{i\in S}\vec{x}^{(i)} \bmod p, \sum_{i\in S} b^{(i)} \bmod p)$ for two independent draws of random subsets S and S', which is $\frac{1}{2^m} + \frac{1}{p^{n+1}}$: indeed, $S = S'$ with probability $\frac{1}{2^m}$, and otherwise there is some i^* that is in one set but not the other – without loss of generality, say $i^* \in S \setminus S'$, and then we have equality iff for all j

$$x_j^{(i^*)} = \sum_{i\in S'} x_j^{(i)} - \sum_{i\in S-i^*} x_j^{(i)} \bmod p \text{ and}$$
$$b^{(i^*)} = \sum i \in S' b^{(i)} - \sum_{i\in S-i^*} b^{(i)} \bmod p$$

which occurs with probability $\frac{1}{p^{n+1}}$. Note furthermore, that if the second draw were an $n+1$-tuple of integers $\{0,\ldots,p-1\}$ chosen uniformly at random, the two draws would be equal with probability $\frac{1}{p^{n+1}}$, which is the second expression; the same holds if both were sampled uniformly at random, giving the third expression. The square is thus equal to $\frac{1}{2^m} + \frac{1}{p^{n+1}} - 2\frac{1}{p^{n+1}} + \frac{1}{p^{n+1}} = \frac{1}{2^m}$. Substituting this value, we obtain that the expected value is equivalent to

$$\frac{\sqrt{p^{n+1}}}{2}\sqrt{\sum_{(\vec{x},r)\in\{0,\ldots,p-1\}^{n+1}}\frac{1}{2^m}} \leq \frac{1}{2p^{n+1}} \text{ since } 2^m \geq p^{5(n+1)}.$$

Noting that the overall expression is nonnegative, we can now invoke Markov's inequality, to obtain that the probability that the advantage is greater than $\frac{1}{\sqrt{2p^{n+1}}}$ is at most $\frac{1}{\sqrt{2p^{n+1}}}$; that is, with probability at least $1 - \frac{1}{\sqrt{2p^{n+1}}}$ over the draw of $\{(\vec{x}^{(i)}, b^{(i)})\}$, the advantage is at most $\frac{1}{\sqrt{2p^{n+1}}}$.

Chapter notes

Occam's razor refers to the following line by William of Ockham from the 14th century:

Numquam ponenda est pluralitas sine necessitate. (Plurality must not be posited without necessity.)

Although he was not the first author to state the principle – essentially the same philosophy appears in Aristotle's *Posterior Analytics* – it has nevertheless been widely attributed to him. The formalization stated in Theorem 4.2 is from Blumer et al. (1987).

Algorithms for learning were proposed long before the modern approaches to their analysis. The Elimination Algorithm, Algorithm 1, was first proposed by Mill (1843, Book III, Chapter 8) for the informally described task of "identifying causes." We noted previously that confidence intervals for statistical estimation appeared in early writing on probability, notably Bernoulli (1713), and the first modern analysis of generalization based on the VC-dimension (which we defer to Appendix A) was carried out by Vapnik and Chervonenkis (1971). The study of the computational complexity of such learning problems was initiated by Valiant (1984). Exercise 4.2 concerns one of many equivalent formalizations of this model, due to Haussler et al. (1991), and is adapted from an exercise in Kearns and Vazirani (1994). Reductions between learning problems were first studied formally by Pitt and Warmuth (1990).

The decision list learning algorithm, Algorithm 2, and its analysis are due to Rivest (1987). The reduction of low-rank decision trees to decision lists (Exercise 4.3) is due to A. Blum (1992), and the exercise is adapted from a similar problem in Kearns and Vazirani (1994). Actually, there is a proper learning algorithm for low-rank decision trees running in essentially the same time, due to Ehrenfeucht and Haussler (1989); the algorithm is a top-down recursive algorithm that is very similar to Algorithm 8 for finding treelike resolution refutations when the tree structure has low rank. Koch, Strassle, and Tan (2023) show that if 3-CNF satisfiability requires exponential time (the "exponential time hypothesis") then decision trees cannot be properly learned in polynomial time. (Indeed, they further propose an additional conjecture that would imply that the running time of the algorithm of Ehrenfeucht and Haussler is optimal.) The question of efficient improper learning of decision trees remains open.

The equivalence of the consistent representation problem with proper learning (in particular its necessity, Theorem 4.11) was proved by Pitt and Valiant (1988). They also observed the NP-completeness of learning 3-term DNF representations (Theorem 4.12), and that weakening the representation to 3CNF yielded a tractable learning problem. It was long believed that improper learning, by contrast, could *not* be NP-complete, and Applebaum, Barak, and Xiao (2008) essentially ruled out *"black-box"* reductions from NP-complete problems to improper learning: that is, reductions that "merely" invoke an algorithm for the problem as a subroutine. Almost all reductions are black-box in this sense. Surprisingly, Hirahara (2022) recently proved that improper learning of programs is NP-complete, indeed using a non-black-box reduction. Nevertheless, this is a very expressive class of functions, and it remains to

be seen whether we can show that problems closer to the boundary of tractable versus intractable are NP-complete.

Cryptography was first observed to pose a barrier to efficient learning in Valiant's original paper (Valiant, 1984): he observed that if *pseudorandom functions* exist (functions indistinguishable from random), Boolean circuits are not efficiently learnable. Pseudorandom functions were first considered by Goldreich, Goldwasser, and Micali (1986), and we now know that their existence is equivalent to the existence of *one-way functions* (Håstad et al., 1999) (i.e., functions that are easy to compute but hard to invert), and in turn to the existence of secure encryption, signature schemes, and many other basic cryptographic constructions (see Goldreich (2001) for an overview). Thus, informally, we may say that learning circuits is impossible if and only if cryptography is possible. But, as we observe (Exercise 4.1), Boolean circuits are essentially the hardest class to learn. Hardness based on the decryption functionality of specific public-key encryption schemes, in the style of Theorem 4.17, was first explored by Kearns and Valiant (1994) to show that improper learning of some much less expressive classes is still intractable.

Problems are usually only presumed to be hard for the purposes of cryptography when they have been "vetted" by generations of mathematicians—in other words, if they are classical problems that still do not have good algorithms. In the case of the shortest vector problem, an algorithm for the problem in two dimensions was first discovered by Lagrange (1773), but is commonly credited to Gauss, who discussed it a few years later; the problem of finding bounds on lattice vectors was a central focus of later work by Minkowski (1896). The extension of Lagrange's algorithm to n dimensions by Lenstra, Lenstra, and Lovasz (1982) only produces a $2^{O(n)}$-approximation. Nevertheless, this algorithm still has played a crucial role in numerous applications in optimization and cryptography. In particular, these algorithms have been used to find approximate integer solutions to equations, such as guaranteed to exist by Dirichlet (1842). Algorithms with slightly better approximation bounds are now known (Schnorr, 1987; Schnorr and Euchner, 1994; Li and Nguyen, 2025), but remain nearly exponential in n, and in particular greater than $2^{n^{1-\epsilon}}$ for all $\epsilon > 0$. At the other end, Khot showed that constant-factor approximations would yield efficient algorithms for NP (Khot, 2005), and Haviv and Regev showed slightly subpolynomial approximations ($2^{(\log n)^{1-\epsilon}}$) yield quasipolynomial-time algorithms for NP (Haviv and Regev, 2007). We remark that Aharanov and Regev showed that the nonexistence of approximations within $\sqrt{n}$ can also be certified, so $\sqrt{n}$-approximations and beyond are unlikely to be NP-hard (this would imply "NP = coNP"). In particular, breaking Regev's encryption only yields $\tilde{O}(n^{3/2})$-approximations, well beyond what could be NP-complete. Regardless, polynomial approximations indeed appear to be well out of reach, in spite of the problem being of strong independent interest among mathematicians. In particular, unlike other famous problems used in cryptography such as integer factoring and discrete logarithms (Shor, 1994), no good *quantum* algorithm is known for the shortest vector problem. Indeed, lattice problems such as GapSVP are currently essentially the only strong candidates for encryption that is secure against

adversaries with quantum computers. Their computational requirements are therefore the subject of intense practical interest.

The first lattice-based encryption scheme was proposed by Ajtai and Dwork (1997), based on a worst-case to average-case reduction for lattice problems due to Ajtai (1996). The public-key encryption scheme described here, as well as an initial analysis based on the Learning With Errors problem, were proposed by Regev (2009). Theorem 4.25, that intersections of halfspaces cannot be improperly learned if Regev's encryption scheme is secure, is due to Klivans and Sherstov (2009). The extension to two-layer neural networks with ReLU or sigmoid activation (Exercise 4.6) was observed by Livni, Shalev-Shwartz, and Shamir (2014).

The reduction from Regev's encryption scheme to the shortest vector problem involved work by many authors. Regev's original reduction used quantum algorithms, and hence relied on the intractability of the shortest vector problem for quantum computers. Peikert (2009) gave the first classical reduction for a slightly different problem ("unique shortest vector"); Lyubashevsky and Miccancio (2009) gave the final piece of the reduction we describe here, via bounded distance decoding, to the classical shortest vector problem, establishing Theorem 4.23. As we noted in the text, our presentation of the reduction to Learning With Errors is incomplete: the main step that we have glossed over is that a sufficiently high-variance Gaussian gives remainders ("modulo p") that are nearly uniformly distributed over the possible values. This was established by Banaszczyk (1993), and the proof is well beyond our scope. The reduction from Learning With Errors to distinguishing structured from random that we describe is from Miccancio and Peikert (2012).

We note briefly that the "Learning With Errors" problem was indeed proposed as a variant of learning a linear function with noisy labels. In this particular case, the hardness of the noisy learning problem can be based on the worst-case hardness of the shortest vector problem. Originally, Blum et al. (1994) had proposed the design of an encryption scheme based on the apparent difficulty of learning parity (exclusive-or) functions with random label noise. So, we observe that the connection between cryptography and learning goes both ways: encryption and related secure computational primitives can be obtained from hard learning problems. The missing piece, that bridges the gap between (not) "learnable" and "indistinguishable from random", is provided by "boosting", presented in Section A.6 of Appendix A.

Between decision trees and intersections of halfspaces, we have DNF formulas. The question of whether DNFs could be efficiently learned was highlighted by Valiant (1984; 1985) as one of the most significant questions in computational learning theory. Daniely and Shalev-Shwartz (2016) proved that if DNF formulas have efficient improper PAC-learning algorithms, then there are algorithms that can distinguish satisfiable k-CNF formulas from k-CNF formulas with $n^{\omega(1)}$ clauses chosen uniformly at random (as a function of k—for example, $n^{\log\log\log\cdots\log k}$ clauses). As a point of reference, we only know how to efficiently refute random k-CNF formulas when there are $\Omega(n^{k/2})$ clauses. This suggests that DNF may not have efficient PAC-learning algorithms. However, compared to the shortest vector problem or integer factoring, the problem of refuting random CNFs has been much less thoroughly explored. As a

cautionary note, the original work by Daniely, Linial, and Shalev-Shwartz (2014), that introduced the techniques used here, had established the same claim under a slightly different assumption that was quickly shown to be false (Allen et al., 2015).

Our introduction to computational learning theory here has focused strictly on supervised learning of Boolean functions in the PAC model. Our focus on Boolean functions in the PAC model is on account of our intent to consider a combined learning and reasoning task, starting in Chapter 6. We return to the question of sample complexity with numerical data in Appendix A, and also discuss the relationship between "weak learning" and the usual, strong PAC learning guarantee, a.k.a. "boosting." But from the standpoint of learning theory more broadly, our coverage is quite incomplete: there are other task formulations, including generative and unsupervised, to name a couple, and other models of learning besides the PAC model. We will briefly introduce and discuss one, mistake-bounded on-line learning, in Section 10.4. We will also briefly discuss learning in the presence of adversarial noise, a.k.a. "agnostic" learning in Section 6.2. But, beyond these, we note that a couple of recent texts cover a much broader range of topics in learning theory: see Shalev-Shwartz and Ben-David (2014) or Mohri, Rostamizadeh, and Talwalkar (2018).

Exercises

4.1 *The most expressive representations.* Prove that if a class C is improperly PAC-learnable by any efficiently evaluatable hypothesis class $\mathcal{H}$, then it is PAC-learnable by Boolean circuits. (***Hint.*** *You may find Proposition 3.26 helpful.*)

4.2 *Representation-size-dependent complexity.* Simple representations such as conjunctions cannot be large relative to the number of attributes, but representations such as decision trees or CNFs could be very large. Often, we would like to allow the number of examples and the running time used by our algorithms to scale with the size of the representation required, but this is not known initially when we run the algorithm. In this problem, we will prove that we can, without loss of generality, assume that we know the size of the target representation. It will be helpful to view our examples as being collected "on demand": rather than a fixed input training set, we suppose the algorithm has a method to obtain a new labeled example of the unknown target property c drawn independently from the unknown distribution D.
Precisely now, suppose we have an algorithm in this model that takes as input n, ϵ, δ, and s where the unknown property c has a representation that is at most s bits long; the algorithm runs in time $T(n, s, 1/\epsilon, 1/\delta)$ for some polynomial T, and with probability $1-\delta$ returns a hypothesis h such that $\Pr_{x\in D}[c(x) = h(x)] \geq 1-\epsilon$. (In particular, it also uses at most $T(n, s, 1/\epsilon, 1/\delta)$ examples.) Prove that there also exists an algorithm that takes only n, ϵ, and δ as input (*not* s), runs in expected time $T'(n, s, 1/\epsilon, 1/\delta)$ for some other polynomial T', and again with probability at least $1-\delta$, returns a hypothesis h such that $\Pr_{x\in D}[c(x) = h(x)] \geq 1-\epsilon$.

4.3 *Decision tree rank versus size.* The *ranks* of the nodes of a rooted binary tree are defined recursively as follows:
- The rank of a leaf is defined to be 0.

- For any internal node of the tree, if both of its children have the same rank r, the node has rank $r+1$. Otherwise, if the children have ranks $r_1 < r_2$, the node also has rank r_2. Finally, the rank of the tree is defined to be the rank of its root node.
(a) Prove that decision trees of rank k on n attributes in which each attribute is read at most once on any path have at least 2^k leaves and at most $n^k + 1$ leaves. (***Hint.*** *Use induction on the rank.*)
(b) Using (a), prove that any decision tree of size S has rank at most $O(\log S)$

4.4 *k-decision lists versus rank k decision trees.* This exercise concerns the state-of-the-art algorithms for learning decision trees.
(a) Recall that decision lists are special cases of decision trees. Consider a decision list with at least one internal node. What is the rank of the decision list when interpreted as a decision tree? Justify your answer.
(b) Recall that we saw that we could represent a k-CNF as a conjunction of attributes corresponding to ORs of literals on at most k of the original n attributes, thus using the Elimination algorithm to learn k-CNFs by learning on this new, enriched set of attributes. We can apply the same trick to decision lists, replacing the literals with ANDs of at most r literals on the original n attributes. We will refer to these as *r-decision lists*.
Prove that the class of functions computed by rank r decision trees is included in the class of functions computed by r-decision lists. Thus, for any fixed r, we can efficiently PAC learn rank r decision trees.
(c) Using (b) and Exercise 4.3(b), prove that decision trees are PAC-learnable in *"quasipolynomial time,"* i.e., size S decision trees on n attributes can be learned in time $2^{\mathrm{poly}(\log n, \log S, \log 1/\epsilon, \log 1/\delta)}$. (You may use the fact that the total number of nodes of a binary tree is always twice the number of leaves minus one.)

4.5 *Decision lists versus halfspaces.*
(a) Prove that any function that can be represented by a decision list can also be represented by a halfspace; and, using Exercise 4.4(b), show that rank-k decision trees can be represented by degree-k polynomial threshold functions.
(b) Give an example of a function that can be represented by a halfspace but cannot be represented by a decision list, and an example of a function that can be represented by a rank-2 decision tree but not by a halfspace. Prove that your examples satisfy these conditions.

4.6 *Sigmoid neural networks are not improperly learnable.*
(a) Prove that two-layer neural networks with sigmoid activation with S hidden units can express the intersection of S halfspaces. In particular, show that if the weights and threshold of the halfspaces are integers, that the magnitude of the largest weight or threshold in hidden units only needs to be $O(\log S)$ times greater than in the halfspaces.
(b) Using (a) and Theorem 4.25, prove that if the approximate Shortest Vector Problem is intractable, two-layer neural networks with sigmoid activation are not efficiently PAC-learnable.