

## 8.1 Polynomial Threshold Functions

In the previous lecture, we proved that any function over  $n$  variables expressible as an  $s$ -term DNF is also expressible as a polynomial threshold function (PTF) of degree  $O(n^{\frac{1}{3}} \log(s))$ , and vice versa. In this lecture, we will present a function which cannot be computed by a PTF of degree  $o(n^{\frac{1}{3}})$ . In other words, we will show that if logarithmic factors are ignored, the bound from the previous lecture is tight. But first, to demonstrate the techniques that will later be used in that proof, we will look at a specific function—the parity function—and prove various things about the degree of its PTF representation.

### 8.1.1 Parity

Let parity :  $\{0, 1\}^n \rightarrow \{0, 1\}$  be the function which is 1 if its input has an odd number of 1's and 0 otherwise. By the following lemma, parity is computed by a degree  $n$  PTF.

**Lemma 1** *Let  $f$  be a Boolean function over  $n$  variables. Then  $f$  is computed by a degree  $n$  PTF.*

**Proof:** For any assignment of the  $n$  variables, there is a degree  $n$  monomial which is 1 on that assignment and 0 otherwise. For example, the monomial

$$x_1 \cdot \bar{x}_2 \cdot \dots \cdot x_{n-1} \cdot \bar{x}_n$$

is 1 on the assignment 10...10 and 0 on every other assignment. Summing over all of the monomials corresponding to assignments  $x$  for which  $f(x) = 1$  yields a degree  $n$  polynomial which is 1 whenever  $f$  is 1 and 0 otherwise. Subtracting  $1/2$  from this and taking the sign of the resulting polynomial yields a PTF of degree  $n$  that computes  $f$ . ■

This lemma gives an upper bound on the required degree of the PTF representation of any Boolean function, including parity. The following theorem gives a lower bound on the degree of any PTF which computes parity.

**Theorem 1** *Parity cannot be computed by a PTF of degree less than  $n$ .*

**Proof:** Let  $\text{sign}(Q(x))$  be a degree  $d$  PTF which computes parity. We are trying to show that  $d$  cannot be less than  $n$ .

Because parity is a symmetric function, for any permutation  $\pi$  of the  $n$  variables,  $\text{sign}(Q(x)) = \text{sign}(Q(\pi(x)))$ . This implies that  $\text{sign}(Q(x) + Q(\pi(x)))$  is also a PTF of degree  $d$

which computes parity. Taken further, consider

$$Q'(x) = \sum_{\pi \in S_n} Q(\pi(x))$$

where  $S_n$  is the set of all permutations of  $\{1, \dots, n\}$ . This process of summing over all possible permutations of the inputs, known as *symmetrization*, yields a symmetric polynomial  $Q'$  of degree  $d$  such that  $\text{sign}(Q') = \text{sign}(Q) = \text{parity}$ . By the following lemma, we can convert this polynomial into a univariate polynomial.

**Lemma 2** *Let  $f(x_1, \dots, x_n)$  be a symmetric polynomial of degree  $m$  over  $\{0, 1\}^n$ . Then there exists a univariate function  $f^*$  of degree  $m$  such that  $f^*(x_1 + \dots + x_n) = f(x_1, \dots, x_n)$ .*

**Proof:** We prove this by showing that every monomial of degree  $k$  has the same coefficient in  $f$ . Because of this, by letting  $c_k$  be the coefficient of the degree  $k$  monomials, we can construct  $f^*$  to be

$$f^*(x) = \sum_{k=0}^m \binom{x}{k} c_k$$

On input  $x$ ,  $f^*(x_1 + \dots + x_n) = f(x_1, \dots, x_n)$  because the equation for  $f^*$  counts, for each  $k$ , the number of clauses of length  $k$  in  $f$  that  $x$  satisfies and multiplies that number by the coefficient that those clauses share. Note that the term of the highest degree in  $f^*$  is

$$\binom{x}{k} = \frac{x \cdot (x-1) \cdot \dots \cdot (x-m+2) \cdot (x-m+1)}{m!}$$

This term has degree  $m$ . Now, we prove using induction that every monomial of degree  $k$  has the same coefficient in  $f$ . For the base case,  $k = 0$ , there is only one monomial. Thus the claim is trivially true in the base case.

For the inductive step, assume that the claim is true for the monomials of degree less than  $k$ . Now, consider two arbitrary monomials of length  $k$ ,  $M_1 = x_{i_1} \cdot x_{i_2} \cdot \dots \cdot x_{i_{k-1}} \cdot x_{i_k}$  and  $M_2 = x_{j_1} \cdot x_{j_2} \cdot \dots \cdot x_{j_{k-1}} \cdot x_{j_k}$ . Consider the input that is 1 for every  $x_i$  in  $M_1$  and 0 otherwise and also the input that is 1 for every  $x_j$  in  $M_2$  and 0 otherwise. Call the first input  $y_i$  and the second  $y_j$ . Both inputs have exactly  $k$  bits that are 1. Because  $f$  is symmetric, it evaluates to the same value on both inputs.

The input  $y_i$  satisfies  $\binom{k}{k-1}$  of the monomials of length  $k-1$ , as does  $y_j$ . By the inductive step, all of these monomials of length  $k-1$  have the same coefficient. In fact, for any integer  $a$  which is less than  $k$ , both  $y_i$  and  $y_j$  satisfy  $\binom{k}{k-a}$  monomials of length  $a$ , all of which, the inductive assumption tells us, have the same coefficient. Thus, the monomials of degree less than  $k$  contribute exactly as much to  $f(y_i)$  as they do to  $f(y_j)$ . The only remaining monomial satisfied by  $y_i$  is  $M_1$ , and the only remaining monomial satisfied by  $y_j$  is  $M_2$ , and because  $f(y_i) = f(y_j)$ ,  $M_1$  must have the same coefficient as  $M_2$ . This proves the lemma. ■

This lemma shows us that because  $Q'$  is a symmetric polynomial of degree  $d$ , we can convert it into a degree  $d$  univariate polynomial  $Q^*$  such that  $Q^*(x_1 + \dots + x_n) = Q'(x_1, \dots, x_n)$ . This, in turn, means that  $\text{parity}(x_1, \dots, x_n) = \text{sign}(Q^*(x_1 + \dots + x_n))$ . Let's analyze the output of  $Q^*$  on various

inputs.

$$\begin{aligned}
 Q^*(0) &< 0 \\
 Q^*(1) &> 0 \\
 Q^*(2) &< 0 \\
 &\vdots \\
 Q^*(2m-1) &> 0 \\
 Q^*(2m) &< 0 \\
 &\vdots
 \end{aligned}$$

$Q^*$  changes its sign at least  $n$  times between 0 and  $n$ , each time passing through 0, meaning it has at least  $n$  roots. Therefore its degree  $d$  is at least  $n$ , and so the degrees of both  $Q'$  and  $Q$  are also at least  $n$ . Thus any PTF which computes parity has degree at least  $n$ . ■

### 8.1.2 One in a Box

Now that we have shown how to lower bound the degree of a PTF which computes the parity function, we use a similar argument to show that for some specifically designed function the degree of a PTF that computes it must be high. This specifically designed function is the Minsky-Papert “One in a Box” function, and it is given in the following definition.

**Definition 1** *The One in a Box function is the function computed by the DNF  $f = T_1 \vee T_2 \vee \dots \vee T_m$ , where each  $T_i = \bigwedge_{j=1}^{4m^2} x_{i,j}$ .*

Each term  $T_i$  is called a box and has its own disjoint set of  $4m^2$  variables, each of which must be 1 for the box to be 1. There are  $m$  such boxes, and so the total number of variables  $n$  is  $4m^3$ . First, we see that because this function is expressible as an  $m$ -term DNF, there is a PTF which computes it of degree  $O(\sqrt[3]{n} \log m) = O(\sqrt[3]{n} \log n)$ . The following theorem shows that we cannot do much better than this.

**Theorem 2** *The “One in a Box” function cannot be computed by a PTF of degree  $o(\sqrt[3]{n})$ .*

**Proof:** Let  $P = \text{sign}(Q)$  be a PTF of degree  $d$  which computes  $f$ . We are trying to show that  $P$  must have a degree at least  $\sqrt[3]{n}$  (specifically,  $\sqrt[3]{n/4}$ ), and since there are  $n = 4m^3$  variables this is the same as showing that the degree of  $P$  is at least  $\sqrt[3]{4m^3/4} = m$ . Because  $P$  is symmetric with respect to each box, we can perform symmetrization on  $Q$ , similar to the one we used when proving the parity lower bound.

Let  $X_i = (x_{i,1}, \dots, x_{i,4m^2})$  and

$$Q' = \sum_{\pi_i \in S_{4m^2}} Q(\pi_1(X_1), \dots, \pi_m(X_m))$$

where again  $S_{4m^2}$  is the set of all permutations of  $\{1, \dots, 4m^2\}$ . As before,  $Q'$  is a boxwise symmetric function over  $0, 1^n$  whose sign equals  $P$ . Therefore, there is some degree  $d$  polynomial  $Q^*$  such that

$$Q^*\left(\sum_{j=1}^{4m^2} x_{1,j}, \dots, \sum_{j=1}^{4m^2} x_{m,j}\right) = Q'(X_1, \dots, X_m)$$

We have that  $Q^* > 0$  exactly when there is a box in which every variable is 1. Since there are  $4m^2$  variables in each box, when one of those summations equals  $4m^2$ ,  $Q^*$  will be positive. Using this fact, let

$$R^*(t) = Q^*(4m^2 - (t-1)^2, 4m^2 - (t-3)^2, \dots, 4m^2 - (t - (2m+1))^2)$$

Note that since we have substituted in terms quadratic in  $t$ ,  $R^*$  has twice the degree of  $Q^*$ . The values of  $t$  that will make at least one of the arguments sent to  $Q^*$  equal to  $4m^2$  are the odd integers between 1 and  $2m+1$ . So, performing a similar analysis as with parity, we have

$$\begin{aligned} R^*(1) &> 0 \\ R^*(2) &< 0 \\ R^*(3) &> 0 \\ &\vdots \\ R^*(2m) &< 0 \\ R^*(2m+1) &> 0 \end{aligned}$$

$R^*$  changes its sign at least  $2m$  times between 1 and  $2m+1$ , so its degree is at least  $2m$ . And since  $R^*$  has twice the degree of  $Q^*$ , the degree of  $Q^*$  is at least  $m$ . The degree of  $Q^*$  equals the degree of  $Q'$ , which in turn equals the degree of  $Q$ . Therefore, the degree of the PTF  $P$  we started with which computes the ‘‘One in a Box’’ function must be at least  $m$ , which is  $\sqrt[3]{n/4}$ . ■

## 8.2 Winnow and its Variants

### 8.2.1 Learning Disjunctions

In a previous lecture we saw how to learn disjunctions over  $n$  variables in running time and mistake bound  $O(n)$ . Now we look at the special case of learning disjunctions of length  $k$ , for some fixed value of  $k$ . For simplicity, we further restrict this to monotone disjunctions. There are roughly  $n^k$  monotone disjunctions of size  $k$ , and we know that disregarding the running time of the learner, we can learn this class with a mistake bound of  $O(k \log(n))$ . What we present in this section is an algorithm which achieves this mistake bound while being restricted to a polynomial (in this case,  $O(n)$ ) running time. The algorithm is a simplified version of the winnow algorithm, and so we will call it the baby winnow algorithm.

**algorithm** Baby Winnow:

1. Begin with the hypothesis  $x_1 + \dots + x_n \geq n$ . The hypothesis will always be of the form  $a_1x_1 + \dots + a_nx_n \geq n$ , where the  $a_i$ 's are weights.
2. For each example  $x$ , perform the following:
  - (a) Evaluate the example using the current hypothesis, and output its guess.
  - (b) If correct, continue.
  - (c) If incorrect:
    - i. If the hypothesis guessed -1, for all  $i$  such that  $x_i = 1$ , double  $a_i$ .
    - ii. Otherwise, for all  $i$  such that  $x_i = 1$ , set  $a_i$  to zero.

It is obvious that the baby winnow algorithm runs in time  $O(n)$ . The following theorem gives us its mistake bound.

**Theorem 3** *The baby winnow algorithm has a mistake bound of  $O(k \log(n))$  when learning disjunctions of length  $k$ .*

**Proof:** Call it a “type 1 mistake” when the hypothesis guesses -1 on an example labelled +1 and likewise a “type 2 mistake” for the other case. We prove this theorem by showing that there can be at most  $k \log(n)$  of either type 1 mistakes or type 2 mistakes.

In the event of a type 1 mistake, the algorithm doubles the weight given to each bit that was set to 1. For example, if the initial hypothesis is presented with the example 10...10, it will output -1, but if the actual label of this example is +1, a mistake will have been made and the hypothesis will be updated to

$$2x_1 + x_2 + \dots + 2x_{n-1} + x_n \geq n$$

A type 1 mistake can only be made when one of the  $k$  variables present in the disjunction we are trying to learn is set to 1. The weight of any of these variables can only be doubled to at most a value of  $n$  because at that point whenever that variable is set to 1, the hypothesis will always predict +1. This means that each of these  $k$  variables can be set to 1 in only  $\log(n)$  type 1 mistakes. But since at least one of these  $k$  variables is set to 1 in every type 1 mistake, at most  $k \log(n)$  type 1 mistakes can be made.

Another property of the update that occurs when a type 1 mistake is made is that it increases the total weight of the halfspace by at most  $n$ . If this were not the case, the total weight of the variables that were set to 1 would have been at least  $n$  and the hypothesis would not have guessed -1. On the other hand, when an update following a type 2 mistake is made, a value of at least  $n$  is removed from the weight of the halfspace. This is because this update zeroes out the weight of every variable set to 1, and the total weight of these variables had to be at least  $n$  for the hypothesis to guess +1.

This removed weight has to come from somewhere, and that somewhere is the increases that occur following type 1 mistakes. Since these updates increase the weight of the halfspace by at most  $n$ , and updates following type 2 mistakes decrease the weight of the halfspace by at least  $n$ , the number of type 2 mistakes is upper bounded by the number of type 1 mistakes. So the number of type 2 mistakes is at most  $k \log(n)$ , meaning that the total number of mistakes is  $O(k \log(n))$ . ■

## 8.2.2 Learning with Experts

Consider a scenario in which you have access to the advice of some expert oracles  $C_1, \dots, C_k$ , each of which when given an example from the instance space outputs a guess of either +1 or -1 depending on what it thinks the example is labeled as. Your goal is to use the advice of these experts in such a way that you are almost as accurate as the best expert. Consider the following algorithm.

**algorithm** Expert Learner

1. Begin with the hypothesis  $C_1 + \dots + C_k \geq 0$ . The hypothesis will always be of the form  $w_1 C_1 + \dots + w_k C_k \geq 0$ , where the  $w_i$ 's are weights.
2. For each example  $x$ , perform the following:
  - (a) Evaluate the example using the current hypothesis, and output its guess.
  - (b) For each incorrect expert  $C_i$ , cut its weight  $w_i$  in half.

By the following theorem, this performs almost as well as the optimal expert.

**Theorem 4** *Let  $opt$  be the number of mistakes made by the best expert. Then this expert learner algorithm makes  $O(opt + \log(k))$  mistakes.*

**Proof:** Let  $W$  be the total weight of the experts. At the beginning,  $W = k$ . When the algorithm makes a mistake, more than half of this weight sided with the wrong guess, and this portion of the weight is halved. So, after the first mistake,  $W$  is reduced from  $k$  to at most  $3/4k$ . Similarly, after  $m$  mistakes,  $W$  is at most  $(\frac{3}{4})^m k$ . We have that  $(\frac{1}{2})^{opt} \leq (\frac{3}{4})^m k$ , where  $opt$  is the number of mistakes made by the best expert. Solving the equation gives us  $m \leq O(opt + \log(k))$ , which proves the bound. ■