On Learning μ-Perceptron Networks On the Uniform Distribution*

Mostefa Golea[†], **Mario Marchand**
Ottawa-Carleton Institute for Physics
University of Ottawa
Ottawa, Ont., Canada K1N 6N5
e-mail: (golea,mario)@physics.uottawa.ca

Thomas R. Hancock
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
e-mail: hancock@learning.scr.siemens.com

October 13, 1995

Acknowledgments: We thank two anonymous referees whose comments and suggestions allowed us to substantially improve the manuscript. M.Marchand is supported by NSERC grant OGP0122405.

Reprint Requests should be sent to Mario Marchand, Physics Department, University of Ottawa, 150 Louis Pasteur, Ottawa, Ontario, Canada K1N 6N5. Phone Number: (613)-564-9293.

Preferred Section: Mathematical and Computational Analysis.

Running Title: **Learning μ-Perceptron Networks**.

Accepted for publication in *Neural Networks*.

*Preliminary versions of some of the results of this paper were presented at the NIPS*92 and EuroColt*93 conferences

[†]Present address: Institute for Social Information Science, Fujitsu Laboratories LTD., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan. email: golea@iias.flab.fujitsu.co.jp.

On Learning μ-Perceptron Networks On the Uniform Distribution

Abstract

We investigate the learnability, under the uniform distribution, of neural concepts that can be represented as simple combinations of *nonoverlapping* perceptrons (also called μ perceptrons) with binary weights and arbitrary thresholds. Two perceptrons are said to be nonoverlapping if they do not share any input variables. Specifically, we investigate, within the distribution-specific PAC model, the learnability of μ *perceptron unions*, *decision lists*, and *generalized decision lists*. In contrast to most neural network learning algorithms, we do not assume that the architecture of the network is known in advance. Rather, it is the task of the algorithm to find both the architecture of the net and the weight values necessary to represent the function to be learned. We give polynomial time algorithms for learning these restricted classes of networks. The algorithms work by estimating various statistical quantities that yield enough information to infer, with high probability, the target concept. Because the algorithms are statistical in nature, they are robust against large amounts of random classification noise.

Key Words: Nonoverlapping Neural Networks, Halfspace Unions, Neural Decision Lists, PAC Learning, Read-Once Formula, Statistical Learning Algorithms, Learning on the Uniform Distribution.

1 Introduction

Within recent years, the field of computational learning theory has emerged to provide a framework for the theoretical understanding of learning algorithms. A central notion in this framework is the "Probably Approximately Correct" (PAC) learning criterion (Valiant, 1984). Loosely speaking, a class F of functions (concepts) is said to be PAC learnable if and only if there exists an *efficient* algorithm such that, when given a *reasonable* number of training examples labeled according to any unknown target function $f \in F$, will *almost always* produce an hypothesis function h that is a *good approximation* of f (see section 3 for a precise definition). Many interesting classes of concepts have been proven to be PAC learnable/unlearnable (Haussler, 1990).

Within the computational learning framework, neural networks have accumulated far more negative (non-learnability) results than positive ones (Blum & Rivest, 1988; Judd, 1988; Kearns & Valiant, 1989; Lin & Vitter, 1991). Even simple neural concept classes such as the intersection of two halfspaces are known to be not learnable under an arbitrary distribution of examples.

The reasonable approach in such situations is to look for positive results by considering restricted classes of the problem, by providing the learning algorithm with additional information in the form of queries, and/or by restricting the distribution generating the examples.

A restriction that has been well studied with respect to boolean formulas is the *read-once* or the μ restriction on specific distributions. A formula is read-once if each variable appears at most once in the formula. In particular, positive learnability results have been obtained for read-once formulas in disjunctive normal form (μ-DNF) on the uniform distribution (Kearns et al., 1987; Pagallo & Haussler, 1989) and read-once boolean formulas whose gates compute functions in {AND,OR,NOT} on product distributions (Schapire, 1991).

Nonoverlapping perceptron networks seem to be the natural neural version of read-once boolean formulas. Two perceptrons are said to be nonoverlapping if they do not share any input variables (Barkai et al., 1990). A μ (or nonoverlapping) perceptron network is a loop-free network in which each node, including the input units has only one outgoing non-zero weight (fig. 1). One can think of this type of architecture as a network of *"decoupled"* perceptrons, which in terms of architecture complexity lies somewhere between the single perceptron and the traditional feedforward neural net. As such, studying this restricted class may shed some light on the gap that exists, in terms of computational complexity, between training single perceptrons, which can be done in polynomial time (Karmarkar, 1984), and training feedforward nets, which has been proven to be intrinsically hard (Blum & Rivest, 1988; Judd, 1988). Of fundamental importance is the question of whether or not removing the overlap between the receptive fields of the nodes makes the learning problem easier.

Standard techniques (Kearns et al., 1987) show that the problem of learning nonoverlapping networks from examples drawn according to an arbitrary distribution is no easier than the problem where the input variables may have an arbitrary number of outgoing weights. Kearns and Valiant (1989) have shown that this general problem is intractable. Kharitonov (1993) has further shown that this intractability result extends to the case of the uniform or indeed any non-trivial input distribution. [1] Thus to obtain positive (learnability) results we must consider a

[1] These hardness results depend on any of several number theoretic conjectures prevalent in cryptography, such as the the difficulty of factoring Blum integers.

slightly easier learning model. For that, we restrict ourselves to the case where the distribution of examples is uniform, and investigate the problem of learning concepts that can be represented as *simple combinations* of *nonoverlapping* perceptrons with binary weights (± 1 or 0) and arbitrary thresholds.

The study of neural networks with binary weights is well motivated from both the theoretical and practical points of view. First, because the number of possible states in the weight space of a network with binary weights is finite, its learning behavior may differ drastically from that of a network with real weights (Watkin et al., 1993). Second, the hardware realization of networks with binary weights may prove to be simpler. The major obstacle impeding the development of binary-weight networks is the lack of efficient learning algorithms. In fact, under an arbitrary distribution of examples, even the simple problem of learning single perceptrons with binary weights is intractable (Pitt and Valiant, 1988). This however does not rule out the existence of efficient learning algorithms that work well under some reasonable distributions, *e.g.* the uniform or product distributions.

Another line of research studies the learnability of μ representations over arbitrary distributions when the learner is allowed to ask membership queries (*i.e.* ask a teacher for the correct classification of a specific instance). In that model a more powerful learner is tackling a harder learning problem. Read-Once boolean formulas (over AND/OR/NOT) are known to be learnable in both models. In another work, we show that arbitrary μ-perceptron networks are learnable with membership queries (Hancock et al., 1994). In this paper, we show that subclasses of μ-perceptron networks are learnable under the uniform distribution. More precisely, we investigate the PAC learnability, under the uniform distribution, of the following concepts when the weights are *binary* valued (see section 2 for precise definitions):

- *μ-perceptron unions*, *i.e.* OR functions of binary perceptrons where each input unit is connected to one and only one perceptron (fig. 2) [2].
- *μ-perceptron decision lists* (fig. 3).
- *Generalized μ-perceptron decision lists* (fig. 4).

Note that in contrast to most neural network learning algorithms, we do not assume that the architecture of the network is known in advance. Rather, it is the task of the algorithm to find both the architecture of the net and the weight values necessary to represent the function to be learned.

We give polynomial time algorithms for learning these special classes of nonoverlapping perceptron networks. The algorithms work by estimating various statistical quantities that yield enough information to infer, with high probability, the target concept. Because of their statistical nature, these algorithms are robust against a large amount of random classification noise (Kearns, 1993).

There are a number of previous algorithms for learning read-once boolean formulas that exploit the idea of using various statistical estimates to infer the target function (Goldman et al., 1990; Pagallo & Haussler, 1989; Schapire, 1991). While the high-level structures of these statistical algorithms are similar, they differ on the the specific statistical tests used. In fact, the main difficulty of applying these techniques is to find the *appropriate* statistical quantities

[2] The intersection is simply the complement of the union and can be treated similarly.

that yield enough information to infer a given class of concepts —a statistical test that works for a particular class of concepts and distributions does not necessarily work for other classes of concepts and/or distributions. In particular, the statistical tests used in the previous algorithms do not apply to nonoverlapping perceptron networks, Thus, while the techniques used in this paper are closely related to the techniques used previously, the specific tests used in this paper (*e.g.* the *effects*, introduced in section 2) are new and non-trivial. Moreover, the class of neural concepts studied here is strictly larger than or incomparable to the function classes studied in the references above. Unlike perceptrons, none of those classes allow nodes that have both unbounded number of inputs and arbitrary thresholds.

2 Definitions

Let $X = \{x_1, x_2..., x_n\}$ be the set of the n input variables and let the input space I^n be the set $\{0,1\}^n$. We assume throughout this paper that the distribution D generating the examples is uniform on I^n.

We denote by $P(A)$ the probability of an event A and by $\hat{P}(A)$ its empirical estimate based on a given finite sample. we denote by $P(A|B)$ the conditional probability of an event A given the fact that event B has been observed. All probabilities are taken with respect to D.

Let f be a boolean function defined on X. The *influence* of a variable x_i on f, denoted $\mathrm{Infl}(x_i)$, is defined as:

$$\mathrm{Infl}(x_i) \stackrel{\mathrm{def}}{=} P(f = 1|x_i = 1) - P(f = 1|x_i = 0).$$

The above expression is a natural measure of the influence of x_i , the degree to which x_i's value affects the value of f (Schapire, 1991). Intuitively, the influence of a variable is positive (negative) if its weight is positive (negative).

For an ordered pair of variables (x_i, x_j) we define the *effect* of x_i on the influence of x_j as

$$\mathrm{Eff}(x_i, x_j) \stackrel{\mathrm{def}}{=} \frac{P(f = 1|x_i = x_j = 1) - P(f = 1|x_i = 1, x_j = 0)}{P(f = 1|x_j = 1) - P(f = 1|x_j = 0)}.$$

Note the the numerator is simply the influence of x_j when x_i is set to 1. Thus, $\mathrm{Eff}(x_i, x_j)$ reflects the effect of setting x_i to 1 on the influence of x_j: setting $x_i = 1$ increases x_j's influence if $\mathrm{Eff}(x_i, x_j) > 1$, decreases x_j's influence if $\mathrm{Eff}(x_i, x_j) < 1$, and has no effect on x_j's influence if $\mathrm{Eff}(x_i, x_j) = 1$. We note that in general, $\mathrm{Eff}(x_i, x_j) \neq \mathrm{Eff}(x_j, x_i)$.

As usual, a perceptron g on X is specified by a vector of n weights w_i and a single threshold θ. For $\mathbf{x} = (x_1, x_2, ..., x_n) \in I^n$, we have:

$$g(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{i=n} w_i x_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{i=n} w_i x_i < \theta \end{cases} \quad (1)$$

We assume without loss of generality that θ is an integer $\in \{-n, \ldots, n+1\}$. A perceptron is said to be monotone (or positive) if $w_i \geq 0$ for $i = 1, ..., n$.

We are interested in the learnability of simple neural concepts built from nonoverlapping perceptrons with binary valued weights ($w_i = \pm 1$ or 0, but, as we will see, this last case is

trivial). For later reference, we define the concepts we shall consider along with some notation and terminology.

Nonoverlapping perceptron unions

A function f is said to be a *perceptron union* if it can be written as a disjunction (OR function) of perceptrons:

$$f = g^{(1)} \vee g^{(2)} \vee ... \vee g^{(r)} \quad 1 \leq r \leq n \tag{2}$$

If the perceptrons do not share any variables, f is said to be a *nonoverlapping* (or μ) *perceptron union* (hereafter μ-PU) (fig. 2).

The *parent* of a variable is the perceptron to which the variable is an immediate input.

We say that two variables are *siblings* if they share a common parent (perceptron), and not siblings otherwise.

Nonoverlapping perceptron decision lists and their generalization

We generalize the notion of *decision lists* (also called linear decision trees), studied by Rivest (1987), to *perceptron decision lists* (hereafter PDLs). A PDL is a list $\mathcal{L}$ of pairs:

$$(g^{(1)}, v^{(1)}), \ldots, (g^{(i)}, v^{(i)}), \ldots, (g^{(r)}, v^{(r)})$$

where each $g^{(i)}$ is a perceptron and $v^{(i)}$ is a value in $\{0, 1\}$. The last perceptron $g^{(r)}$ is the constant function $+1$. This defines a function $f : I^n \rightarrow \{0, 1\}$ as follows: for any $\mathbf{x}$, $f(\mathbf{x})$ is defined to be equal to $v^{(j)}$ where j is the least index for which $g^{(j)}(\mathbf{x}) = 1$.

As in (Rivest, 1987), we may think of a PDL as an extended "**if-then-elseif-...else-**" rule. Compared to Rivest's decision lists, PDLs have the same structure but the complexity of the decision allowed at each node is greater.

A PDL can be converted to an equivalent feedforward neural net in an obvious manner (see fig. 3a and b). The result is a network of *alternating levels* of disjunction (OR) and conjunction (AND) of perceptrons (fig. 3c).

A possible generalization of PDL is to allow multiple AND/OR nodes at each level (fig. 4). Such networks represent arbitrary boolean formulas over the functions {AND,OR} whose inputs are perceptrons. We call such a network (function) a *generalized* PDL.

If each variable appears at most once, the PDL is referred to as *nonoverlapping* (or μ) *perceptron decision list* (hereafter μ-PDL). Similarly, we refer to generalized PDLs with the same property as *generalized* μ-PDLs. The equivalent network can be then viewed as a rooted tree. The root is the *output node* and each leaf in the tree is labeled with an input variable in a way such that no variable appears on more than one leaf. We will refer to the AND/OR nodes as *gates*. We will say that input x_i (or a perceptron g or a gate Γ) *feeds* a gate Γ' if the path from x_i (or g or Γ) to the output goes through Γ'.

We will be interested in learning μ-PDL and generalized μ-PDLs with binary (± 1, or 0) weights and arbitrary thresholds. To simplify the analysis, we assume w.l.o.g. that the input variables feed the gates only through perceptrons (maybe single-variable perceptrons) and that each gate has, as input, at least one multi-variable perceptron or another gate (otherwise, it is a perceptron).

The *parent* of a variable is the perceptron to which the variable is an immediate input. We say that a perceptron g (or a gate Γ) is an *uncle* of a variable x_i if g (respectively, Γ) is an immediate input to a gate fed by x_i, but g (respectively, Γ) is not itself fed by x_i. Hence, only perceptrons can be an uncle of a variable belonging to a decision list whereas both perceptrons and gates can be an uncle of a variable belonging to a generalized decision list.

The *depth* of a gate Γ (or a variable x_i or a perceptron g) is the number of gates on the path from Γ (or x_i or g) to the output gate. We say that x_i is a *zero-level* variable *with respect to* a gate Γ if x_i's parent is an immediate input to Γ.

The *least common ancestor* of two variables x_i and x_j, denoted lca(x_i,x_j), is the deepest gate fed by both x_i and x_j. We say that two variables are *siblings* if they share a common parent (perceptron), and not siblings otherwise.

Some of the definitions given here are illustrated in fig. 4.

3 The PAC Learning Criteria

In what follows, we adopt the PAC criteria for learning (Valiant, 1984), restricted to the uniform distribution. Here the methodology is to draw a sample of a certain size labeled according to the unknown target function f and then to find a "good" approximation h of f. The error of the hypothesis function h, with respect to the target f, is defined to be

$$P(h \neq f) \stackrel{\text{def}}{=} \Pr_{\mathbf{x} \in D} \{h(\mathbf{x}) \neq f(\mathbf{x})\}$$

where $\mathbf{x}$ is drawn according to the *same* distribution D used to generate the training sample.

An algorithm learns from examples a target class F under the distribution D on I^n, if for every $f \in F$, and any $0 < \epsilon, \delta < 1$, the algorithm runs in time polynomial in $(n, 1/\epsilon, 1/\delta)$ and outputs an hypothesis h such that, with probability at least $1 - \delta$,

$$P(h \neq f) \ < \ \epsilon$$

4 The Learning Algorithms

In order to simplify the analysis, we introduce the following notation. Let N be the number of negative weights in g and let $\mathbf{y}$ be defined as

$$y_i = \begin{cases} x_i & \text{if } \ w_i = 1 \text{ or } 0 \\ 1 - x_i & \text{if } \ w_i = -1 \end{cases} \tag{3}$$

Then eq. 1 can be written as

$$g(\mathbf{y}) = \begin{cases} 1 & \text{if } \ \sum_{i=1}^{i=n} y_i \ \geq \ \Omega \\ 0 & \text{if } \ \sum_{i=1}^{i=n} y_i \ < \ \Omega \end{cases} \tag{4}$$

where the renormalized threshold Ω is related to the original threshold θ by: $\Omega = \theta + N$. We assume, w.l.o.g., that $0 \leq \Omega \leq n+1$. Note that if $\mathbf{x}$ is uniformly distributed on I^n, then so is $\mathbf{y}$.

4.1 Preliminary Lemmas

We will make use of the following lemmas.

Lemma 1 *Let d and n be two integers. Then, if $\frac{n}{2} \leq d \leq n$,*

$$\sum_{i=d}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) \leq \frac{1}{2} \left(\begin{array}{c} n \\ d \end{array} \right) \frac{1}{1-z}$$

where $z = \frac{1}{2}\frac{n+1}{d+1}$. And if $0 \leq d \leq \frac{n}{2}$,

$$\sum_{i=0}^{d} \left(\begin{array}{c} n \\ i \end{array} \right) \leq \frac{1}{2} \left(\begin{array}{c} n \\ d \end{array} \right) \frac{1}{1-z'}$$

where $z' = \frac{1}{2}\frac{n+1}{n-d+1}$.

Proof: follows directly from Bahadur's expansion (Bahadur, 1960).□
This lemma is used throughout this paper to approximate binomial distributions.

Lemma 2 *Let S be the number of successes in m Bernoulli trials each with probability of success p (and probability of failure $q = 1 - p$). For any $\epsilon \geq 0$,*

$$\Pr\left(\left|\frac{S}{m} - p\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{\epsilon^2 m}{4pq}\right)$$

Proof: follows directly from Chernoff bounds. See for example (Cormen et al., 1990, Corollary 6.7).□
Note that $\frac{S}{m} = \hat{p}$, where $\hat{p}$ is the empirical estimate of p obtained from m trials. Thus, if we want to ensure that $|\hat{p} - p| \leq \epsilon$ with probability at least $1 - \delta$, it suffices to pick m such that

$$2\exp\left(-\frac{\epsilon^2 m}{4pq}\right) \leq \delta$$

which is true if

$$m \geq \frac{1}{\epsilon^2} \ln \frac{2}{\delta}$$

as $4pq \leq 1$. This last expression will be used throughout the paper to determine the number of examples necessary to approximate a given probability within a given accuracy.

Lemma 3 *Let g be a perceptron with binary weights such that $P(g = 1)$, $P(g = 0) > \rho$, where $0 < \rho \leq 1/2$. Then,*

$$P(g(\mathbf{x}) = 1 | x_i = 1) - P(g(\mathbf{x}) = 1 | x_i = 0) \begin{cases} > & +\frac{\rho}{2(n+2)} & \text{if } w_i = +1 \\ = & 0 & \text{if } w_i = 0 \\ < & -\frac{\rho}{2(n+2)} & \text{if } w_i = -1 \end{cases}$$

Proof: We first note that from the definition of the influence and eq. 3, we can write:

$$P(g(\mathbf{x})=1|x_i=1)-P(g(\mathbf{x})=1|x_i=0)=\begin{cases} +(P(g(\mathbf{y})=1|y_i=1)-P(g(\mathbf{y})=1|y_i=0)) & \text{if } w_i=+1 \\ -(P(g(\mathbf{y})=1|y_i=1)+P(g(\mathbf{y})=1|y_i=0)) & \text{if } w_i=-1 \\ 0 & \text{if } w_i=0 \end{cases}$$

Focusing on the cases $w_i \neq 0$, we exploit the independence of the input variables to write

$$\begin{aligned} P(g(\mathbf{y})=1|y_i=1)-P(g(\mathbf{y})=1|y_i=0) &= P(\sum_{j\neq i} y_j = \Omega-1) = \left(\begin{array}{c} n-1 \\ \Omega-1 \end{array}\right)/2^{n-1} \\ &= 2\frac{\left(\begin{array}{c} n-1 \\ \Omega-1 \end{array}\right)}{\sum_{i=\Omega}^{n}\left(\begin{array}{c} n \\ i \end{array}\right)}P(g=1) \qquad (5) \\ &= 2\frac{\left(\begin{array}{c} n-1 \\ \Omega-1 \end{array}\right)}{\sum_{i=0}^{\Omega-1}\left(\begin{array}{c} n \\ i \end{array}\right)}P(g=0) \qquad (6) \end{aligned}$$

Eqs. 5 and 6 follow from the fact that, under the uniform distribution,

$$P(g=1)=\sum_{i=\Omega}^{n}\left(\begin{array}{c} n \\ i \end{array}\right)/2^n$$

and

$$P(g=0)=\sum_{i=0}^{\Omega-1}\left(\begin{array}{c} n \\ i \end{array}\right)/2^n$$

respectively.

Case 1: $\Omega \geq \frac{n}{2}$
Using eq. (5) and lemma 1, it is easy to see that

$$\begin{aligned} P(g(\mathbf{y})=1|y_i=1)-P(g(\mathbf{y})=1|y_i=0) &\geq \frac{\left(\begin{array}{c} n-1 \\ \Omega-1 \end{array}\right)}{\frac{1}{2}\left(\begin{array}{c} n \\ \Omega \end{array}\right)}\left(1-\frac{1}{2}\frac{n+1}{\Omega+1}\right)P(g=1) \\ &= \frac{2\Omega}{n}\left(1-\frac{1}{2}\frac{n+1}{\Omega+1}\right)P(g=1) \\ &> \frac{\rho}{2(n+2)} \end{aligned}$$

Case 2: $\Omega \leq \frac{n}{2}$
Using eq. (6) and lemma 1, it is again easy to see that

$$P(g(\mathbf{y})=1|y_i=1)-P(g(\mathbf{y})=1|y_i=0) \geq \frac{\left(\begin{array}{c} n-1 \\ \Omega-1 \end{array}\right)}{\frac{1}{2}\left(\begin{array}{c} n \\ \Omega-1 \end{array}\right)}\left(1-\frac{1}{2}\frac{n+1}{n-\Omega+2}\right)P(g=0)$$

$$\begin{aligned} &= \frac{2(n-\Omega+1)}{n}\left(1-\frac{1}{2}\frac{n+1}{n-\Omega+2}\right)P(g=0) \\ &> \frac{\rho}{2(n+2)} \end{aligned}$$

□.

4.2 Learning μ-Perceptron Unions

In this section, we describe a polynomial time algorithm for learning μ-PUs with binary weights from random examples drawn according to the uniform distribution D. Note that we do not assume that the architecture (i.e. the connectivity and number of perceptrons) is known in advance. Rather, it is the task of the learning algorithm to determine which variables are in a given perceptron.

Let us assume that the target function f is a μ-PU as in eq. 2. We shall assume w.l.o.g. that f is expressed with the maximum possible number of $g^{(i)}$'s, i.e. an OR of variables is represented as the OR of individual perceptrons. For example, if $f = g^{(1)} \vee g^{(2)} \vee g^{(2)}$ and $g^{(1)} = x_1 \vee x_2 \vee x_3$, then f is expressed as

$$f = g_1^{(1)} \vee g_2^{(1)} \vee g_3^{(1)} \vee g^{(2)} \vee g^{(2)}$$

where $g_i^{(1)} = x_i$ $(i = 1, 2, 3)$.

The learning algorithm proceeds in three steps:

1. In the first step, the algorithm determines the "relevant" input variables and the values (signs) of their weights. To achieve this, for each variable x_i, the algorithm estimates its *influence* $\mathrm{Infl}(x_i)$ using examples drawn randomly according to D. We prove that if the variable x_i is relevant, then its influence is significantly greater or less than zero, depending on whether $w_i = 1$ or $w_i = -1$. If $|\mathrm{Infl}(x_i)|$ is too small, the algorithm concludes that the variable is "irrelevant" and so, neglects it in later stages. Once the weights are estimated, the algorithm reduces the target function to a monotone μ-PU by simply changing x_i to $1 - x_i$ whenever $w_i = -1$.

2. In the second step, the algorithm determines which variables are siblings, *i.e.* belong to the same perceptron. To do this, the algorithm estimates the *effect* $\mathrm{Eff}(x_i, x_j)$ for each ordered pair of variables using examples drawn randomly according to D. We show below that these *effects* contain enough information to infer the architecture of the network.

3. In the last step, the algorithm estimates the threshold value for each perceptron.

Intuition suggests that the influence of a variable is positive (negative) if its weight is positive (negative). The following lemma strengthens this intuition by showing that there is a measurable *gap* between the two cases. This gap will be used to estimate the weight values (signs).

Lemma 4 *Let f be a μ-perceptron union as in eq.2. Let g be a perceptron in f and let $x_i \in g$. Assume that $P(f = 1) < 1 - \gamma$ and $P(g = 1), P(g = 0) > \rho$ where $0 < \gamma,\ \rho < 1$. Then*

$$\mathrm{Infl}(x_i) \begin{cases} > & +\dfrac{\gamma\rho}{2(n+2)} & \text{if} \quad w_i = +1 \\ = & 0 & \text{if} \quad w_i = 0 \\ < & -\dfrac{\gamma\rho}{2(n+2)} & \text{if} \quad w_i = -1 \end{cases}$$

Proof: We first note that $\mathrm{Infl}(x_i) = 0$ if $w_i = 0$. Also note that $\mathrm{Infl}(x_i) = \mathrm{Infl}(y_i)$ if $w_i = +1$ and $\mathrm{Infl}(x_i) = -\mathrm{Infl}(y_i)$ if $w_i = -1$. Let m be the disjunction of all perceptrons in f except g. Then, using the inclusion-exclusion property and the fact that perceptrons in f do not share any variables:

$$\begin{aligned}
\mathrm{Infl}(y_i) &= (1 - P(m = 1))(P(g = 1|y_i = 1) - (P(g = 1|y_i = 0)) \\
&> \gamma(P(g = 1|x_i = 1) - (P(g = 1|x_i = 0)) \qquad (7) \\
&> \frac{\gamma\rho}{2(n+2)} \qquad (8)
\end{aligned}$$

Inequality 7 follows from the fact that $1 - P(m = 1) > 1 - P(f = 1) > \gamma$, and inequality 8 follows from lemma 3. □

Note that we can assume w.l.o.g. that $\gamma \geq \epsilon/2$, for otherwise we can set f to the constant function 1 without introducing much error. Likewise, we can assume w.l.o.g. that $\rho \geq \epsilon/2n$, for otherwise we can neglect perceptron g without introducing much error (see sketch of the algorithm in figure 5 and proof of theorem 6). Under the above assumptions, the gap is wide enough to be estimated efficiently using a sample of polynomial size (lemma 2).

Lemma 4 enables us to determine the weight values and reduce f to its monotone form. The next step is to determine which variables belong to the same perceptron. Starting with a variable, say x_i, the algorithm uses the effect measure to decide whether or not another variable, say x_j, belongs to x_i's parent. We appeal to the following lemma where we assume that f is already reduced to its monotone form.

Lemma 5 *Let f be a μ-perceptron union (monotone form). Let $g^{(a)}$ be a perceptron in f. Let $x_i \in g^{(a)}$ and let x_j and x_k be two other influential variables in f. Then*

$$\mathrm{Eff}(x_i, x_j) - \mathrm{Eff}(x_i, x_k) = \begin{cases} 0 & \text{if } x_j \in g^{(a)} \text{ and } x_k \in g^{(a)} \\ 0 & \text{if } x_j \notin g^{(a)} \text{ and } x_k \notin g^{(a)} \\ \geq \frac{1}{n^2} & \text{if } x_j \in g^{(a)} \text{ and } x_k \notin g^{(a)} \end{cases}$$

Proof: It is easy to see that if $x_j, x_k \in g^{(a)}$, then $\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}(x_i, x_k)$. Now, assume that $x_k \notin g^{(a)}$, say $x_k \in g^{(b)}$, and let m be the disjunction of all perceptrons in f except $g^{(a)}$ and $g^{(b)}$. Using the the inclusion-exclusion property and the fact that perceptrons in f do not share any variables, we get

$$\begin{aligned}
\mathrm{Eff}(x_i, x_k) &\stackrel{\text{def}}{=} \frac{P(f = 1|x_i = x_k = 1) - P(f = 1|x_i = 1, x_k = 0)}{P(f = 1|x_k = 1) - P(f = 1|x_k = 0)} \\
&= \frac{(1 - P(m = 1))\left(1 - P(g^{(a)} = 1|x_i = 1)\right)\left(P(g^{(b)} = 1|x_k = 1) - P(g^{(b)} = 1|x_k = 0)\right)}{(1 - P(m = 1))\,(1 - P(g^{(a)} = 1))\,(P(g^{(b)} = 1|x_k = 1) - P(g^{(b)} = 1|x_k = 0))} \\
&= \frac{1 - P(g^{(a)} = 1|x_i = 1)}{1 - P(g^{(a)} = 1)} = 1 - \frac{P(g^{(a)} = 1|x_i = 1) - P(g^{(a)} = 1)}{1 - P(g^{(a)} = 1)} \stackrel{\text{def}}{=} \mathrm{Eff}_{II} \qquad (9)
\end{aligned}$$

which is independent of x_k. So, if x_j, $x_k \notin g^{(a)}$, then $\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}(x_i, x_k)$. We are left with the case where $x_j \in g^{(a)}$ and $x_k \notin g^{(a)}$. In this case, $\mathrm{Eff}(x_i, x_k)$ is given by eq. (9), and $\mathrm{Eff}(x_i, x_j)$

is given by

$$\begin{aligned}
\mathrm{Eff}(x_i,x_j) &\stackrel{\mathrm{def}}{=} \frac{P(f=1|x_i=x_j=1)-P(f=1|x_i=1,x_j=0)}{P(f=1|x_j=1)-P(f=1|x_j=0)} \\
&= \frac{\Big(1-P(g^{(b)}=1)\Big)\Big(P(g^{(a)}=1|x_i=x_j=1)-P(g^{(a)}=1|x_i=1,x_j=0)\Big)}{(1-P(g^{(b)}=1))\,(P(g^{(a)}=1|x_j=1)-P(g^{(a)}=1|x_j=))} \\
&= \frac{P(g^{(a)}=1|x_i=x_j=1)-P(g^{(a)}=1|x_i=1,x_j=0)}{P(g^{(a)}=1|x_j=1)-P(g^{(a)}=1|x_j=0)} \stackrel{\mathrm{def}}{=} \mathrm{Eff}_I
\end{aligned} \tag{10}$$

Let p be the number of variables in $g^{(a)}$ and let v be its renormalized threshold. Then, under the uniform distribution:

$$1-P(g^{(a)}=1) \equiv P(g^{(a)}=0) = \sum_{i=0}^{v-1} \left(\begin{array}{c} p \\ i \end{array} \right) /2^p$$

$$P(g^{(a)}=1|x_j=1)-P(g^{(a)}=1) = \frac{1}{2}P(g^{(a)}=1|x_j=1)-\frac{1}{2}P(g^{(a)}=1|x_j=0) = \left(\begin{array}{c} p-1 \\ v-1 \end{array} \right) /2^{p-1}$$

$$P(g^{(a)}=1|x_i=x_j=1)-P(g^{(a)}=1|x_i=1,x_j=0) = \left(\begin{array}{c} p-2 \\ v-2 \end{array} \right) /2^{p-2}$$

Hence

$$\mathrm{Eff}(x_i,x_j) = \mathrm{Eff}_I = 2\frac{v-1}{p-1} \tag{11}$$

$$\mathrm{Eff}(x_i,x_k) = \mathrm{Eff}_{II} = \frac{\left(\begin{array}{c} p-1 \\ v-1 \end{array} \right)}{\sum_{i=0}^{v-1} \left(\begin{array}{c} p \\ i \end{array} \right)} - 1 \tag{12}$$

Case 1: $p \geq v \geq \frac{p}{2}+1$

$$\mathrm{Eff}(x_i,x_j) - \mathrm{Eff}(x_i,x_k) \geq 2\frac{v-1}{p-1} - 1 \geq \frac{1}{p-1} \geq \frac{1}{n-1} \geq \frac{1}{n^2}$$

Case 2: $2 \leq v \leq \frac{p}{2}$

In this case, applying lemma 1 to $\sum_{i=0}^{v-1} \left(\begin{array}{c} p \\ i \end{array} \right)$ we get

$$\begin{aligned}
\mathrm{Eff}(x_i,x_j) - \mathrm{Eff}(x_i,x_k) &\geq 2\frac{v-1}{p-1} + 2\left(\frac{p-v+1}{p}\right)\left(1-\frac{1}{2}\frac{p+1}{p-v+2}\right) - 1 \\
&\geq \frac{3p-1}{p^2(p-1)} \geq \frac{1}{p^2} \geq \frac{1}{n^2}
\end{aligned} \tag{13}$$

Note that v is different from 1, that is $g^{(a)}$ is not a perceptron that computes an OR function (otherwise, x_i would have to be the only input to $g^{(a)}$, contracting the fact that both x_i and x_j are inputs to $g^{(a)}$).□

If we estimate the *effects* to within a precision better than the *effect* gap, established in lemma 5, we will be able to decide which variables are in the same perceptron.

The last step of the algorithm is to estimate the threshold of each perceptron g. Two cases are possible:

case 1: g is a single-variable perceptron. Then the only possible threshold value is 1.

case 2: g is a multi-variable perceptron. Let p be the number of variables in g and let v be its renormalized threshold. Let x_i, $x_j \in g$. Then (eq. 10)

$$\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}_I = 2\frac{v-1}{p-1}$$

Thus, a good estimation of $\mathrm{Eff}(x_i, x_j)$ yields a good estimation of the renormalized threshold v (and hence the original threshold).

A sketch of the algorithm for learning μ-perceptron unions is given in fig. 5.

Theorem 6 *The class of μ-perceptron unions with binary weights is PAC learnable under the uniform distribution.*

Proof: First, we claim that a sample of size $m = (80)^4 \left(\frac{n^6(n+2)^2}{\epsilon^4}\right) \ln \frac{16n^2}{\delta}$ is sufficient to ensure that the different probabilities are estimated to within a sufficient precision, i.e.

1. $|\hat{P}(f=1) - P(f=1)| < \epsilon/4$ with confidence at least $1 - \delta/2$.
2. $|\hat{Infl}(x_i) - \mathrm{Infl}(x_i)| < \frac{\epsilon^2}{16n(n+2)}$ with confidence at least $1 - \delta/8n$, for $i = 1, \ldots, n$.
3. $|\hat{\mathrm{Eff}}(x_i, x_j) - \mathrm{Eff}(x_i, x_j)| < \frac{1}{8n^2}$ with confidence at least $1 - \delta/4n^2$, for $i, j = 1, \ldots, n$ and $i \neq j$.

The first claim follows from Chernoff bounds (lemma 2). We prove the second claim; the third one follows by a similar argument. Recall that

$$\mathrm{Infl}(x_i) = P(f=1|x_i=1) - P(f=1|x_i=0)$$

To show that

$$|\hat{Infl}(x_i) - \mathrm{Infl}(x_i)| \leq \frac{\epsilon^2}{16n(n+2)}$$

we only need to establish that

$$|\hat{P}(f=1|x_i=1) - P(f=1|x_i=1)| \leq \frac{1}{2}\frac{\epsilon^2}{16n(n+2)}$$

and

$$|\hat{P}(f=1|x_i=0) - P(f=1|x_i=0)| \leq \frac{1}{2}\frac{\epsilon^2}{16n(n+2)}$$

For this, we express the conditional probabilities in terms of probabilities that are estimated by using *all* the training examples. For $c \in \{0, 1\}$,

$$P(f = 1|x_i = c) = \frac{P(f = 1 \ \& \ x_i = c)}{P(x_i = c)}$$

Let $\epsilon' = \frac{1}{320}\frac{\epsilon^2}{n(n+2)}$. Then. lemma 2 yields

$$|\hat{P}(f = 1 \ \& \ x_i = c) - P(f = 1 \ \& \ x_i = c)| \leq \epsilon'$$

and

$$|\hat{P}(x_i = c) - P(x_i = c)| \leq \epsilon'$$

with very high probability. Now

$$\begin{aligned}
\hat{P}(f = 1|x_i = c) &= \frac{\hat{P}(f = 1 \ \& \ x_i = c)}{\hat{P}(x_i = c)} \\
&\leq \frac{P(f = 1 \ \& \ x_i = c) + \epsilon'}{P(x_i = c) - \epsilon'} \\
&= (\frac{P(f = 1 \ \& \ x_i = c)}{P(x_i = c)} + \frac{\epsilon'}{P(x_i = c)})(\frac{1}{1 - \ \epsilon'/P(x_i = c)} \\
&= (P(f = 1|x_i = c) + \beta)\frac{1}{1 - \beta}
\end{aligned}$$

where $\beta = \frac{\epsilon'}{P(x_i=c)}$. Since $P(x_i = c) = 1/2$ (uniform distribution), $\beta = 2\epsilon' < 1/2$.Thus, $(1 - \beta)^{-1} < 1 + 2\beta$. Using these inequalities in the above expression, we get

$$\begin{aligned}
\hat{P}(f = 1|x_i = c) &< P(f = 1|x_i = c) + \beta)(1 + 2\beta) \\
&= P(f = 1|x_i = c) + \beta + 2\beta P(f = 1|x_i = c) + 2\beta^2 \\
&< P(f = 1|x_i = c) + 5\beta = P(f = 1|x_i = c) + 5\frac{\epsilon'}{P(x_i = c)} \\
&= P(f = 1|x_i = c) + \frac{1}{32}\frac{\epsilon^2}{n(n + 2)} \qquad (14)
\end{aligned}$$

Similarly,

$$\hat{P}(f = 1|x_i = c) \geq \frac{P(f = 1 \ \& \ x_i = c) - \epsilon'}{P(x_i = c) + \epsilon'} = (P(f = 1|x_i = c) - \beta)\frac{1}{1 + \beta}$$

Since $(1 + \beta)^{-1} < 1 - \beta$, it follows that:

$$\begin{aligned}
\hat{P}(f = 1|x_i = c) &\geq (P(f = 1|x_i = c) - \beta)(1 - \beta) \\
&> P(f = 1|x_i = c) - 2\beta > (P(f = 1|x_i = c) - \frac{1}{32}\frac{\epsilon^2}{n(n + 2)} \qquad (15)
\end{aligned}$$

Combining 14 and 15 yields the desired result.

Note that the effects and influences are estimated within precisions that are smaller than the gaps established in lemmas 4 and 5. Hence, the algorithm is able to identify, with high probability, the weight values and the set of siblings for each variable.

The rest of the analysis is straightforward:

- If the algorithm outputs $h = 1$ (step 2), then

$$\begin{aligned} P\left(h \neq f\right) \leq P(f=0)) &< \hat{P}(f=0) + \epsilon/4 \\ &\leq 3\epsilon/4 + \epsilon/4 = \epsilon \end{aligned} \tag{16}$$

- If the algorithm outputs $h = 0$ (step 3), then

$$\begin{aligned} P\left(h \neq f\right) \leq P(f=1)) &< \hat{P}(f=1) + \epsilon/4 \\ &\leq 3\epsilon/4 + \epsilon/4 = \epsilon \end{aligned}$$

- If we reach step 4, then

$$P(f=1) > \hat{P}(f=1) - \epsilon/4 > 3\epsilon/4 - \epsilon/4 = \epsilon/2$$

  and

$$P(f=0) > \hat{P}(f=0) - \epsilon/4 > 3\epsilon/4 - \epsilon/4 = \epsilon/2$$

  Thus, as we said in the proof of lemma 4, we may put $\gamma = \frac{\epsilon}{2}$, w.l.o.g.

- In step 5, we choose to ignore any variable $x_i \in g^{(a)}$ for which $P(g^{(a)} = 1) < \rho = \frac{\epsilon}{2n}$. We argue that ignoring such variables (and hence ignoring their parent perceptrons) will not introduce much error in the final hypothesis. To see this, let f' be the function obtained from f by ignoring k perceptrons $g^{(1)}, g^{(2)}, \ldots, g^{(k)}$. It is not hard to see that

$$P(f \neq f') \leq \sum_{i=1}^{k} P(g^{(i)})$$

  If $P(g^{(i)} = 1) < \rho = \epsilon/2n$, for $i = 1, \ldots, k$, then $P(f \neq f') < k \times \rho < \epsilon$.
  Hence, the hypothesis h output in step 6a will have an error less than ϵ with respect to the target function.

Finally, it takes $O(m)$ units of time to estimate a probability using a sample of size m. Thus, it takes $O(m \times n^2)$ units of times to estimate all the influences and effects. Further, it takes $O(n)$ units of time to determine the siblings of each variable and $O(1)$ units time to determine the threshold of each perceptron. Hence, the running time of the algorithm is $O(m \times n^2)$.□

We note in passing that Chernoff bounds only provide a crude lower bound on the sample complexity. In practice, learning algorithms require much fewer examples (Golea, 1994). Hence in the following, we will give only the *order* of the number of examples needed for PAC learnability.

4.3 Learning μ-Perceptron Decision Lists

In this section, we describe a polynomial time algorithm for learning μ-PDLs with binary weights from random examples drawn according to the uniform distribution D. Actually, what we will be learning is the equivalent network of the μ-PDL (fig. 3). The next section will provide the necessary additional processing to handle generalized μ-PDLs.

We assume w.l.o.g. that the target network contains no two successive AND or two successive OR gates, as such nodes can always be merged. We also assume w.l.o.g. that the target network has the maximum possible number of perceptrons.

The learning algorithm proceeds in three steps:

1. In the first step, the algorithm determines the "relevant" input variables and the values (signs) of their weights. To achieve this, for each variable x_i, the algorithm estimates its *influence* $\mathrm{Infl}(x_i)$ using examples drawn randomly according to D. We prove that if the variable x_i is relevant, then its influence is significantly greater or less than zero, depending on whether $w_i = 1$ or $w_i = -1$. If $|\mathrm{Infl}(x_i)|$ is too small, the algorithm concludes that the variable is "irrelevant" and so, neglects it in later stages. Once the weights are estimated, the algorithm reduces the target function to a monotone μ-PDL by simply changing x_i to $1 - x_i$ whenever $w_i = -1$.

2. In the second step, the algorithm infers the architecture of the network (fig. 3), *i.e.* determines which variables appear in a given level, and among those, which are siblings. To do this, the algorithm estimates the effect $\mathrm{Eff}(x_i, x_j)$ for each ordered pair of variables using examples drawn randomly according to D. We show that these effects contain enough information to infer the architecture of the network.

3. In the last step, the algorithm estimates a threshold value for each perceptron. We will see below that the effects provide enough information to do that.

The following lemma shows that there is a measurable *gap* in the influence of a variable, between the two cases $w_i = 1$ and $w_i = -1$. Hence, this gap is used to estimate the weight value (sign) emerging from each variable.

Lemma 7 *Let f be μ-PDL. Let g be a perceptron in f and let $x_i \in g$. Let $\{\lambda^{(s)}\}$ be the set of x_i's uncles and let $\{f^{(s)}\}$ be the set of subformulas computed by those uncles. Let $a^{(s)} = 1$ if $\lambda^{(s)}$ feeds immediately an AND gate and $a^{(s)} = 0$ if $\lambda^{(s)}$ feeds immediately an OR gate. Then, if $P(g = 1)$, $P(g = 0) > \rho$ and $\prod_s P(f^{(s)} = a^{(s)}) > \gamma$,*

$$\mathrm{Infl}(x_i) \begin{cases} > & +\frac{\gamma\rho}{2(n+2)} & \text{if} \quad w_i = +1 \\ = & 0 & \text{if} \quad w_i = 0 \\ < & -\frac{\gamma\rho}{2(n+2)} & \text{if} \quad w_i = -1 \end{cases}$$

Proof: it is easy to show by induction on the depth of f that

$$\mathrm{Infl}(x_i) = (P(g = 1|x_i = 1) - P(g = 1|x_i = 0)) \prod_s P(f^{(s)} = a^{(s)})$$

The result then follows then from the conditions of this lemma and lemma 3. $\Box$.

Note that we can assume w.l.o.g. that $\rho \geq \epsilon/2n$, for otherwise we can neglect perceptron g without introducing much error. Likewise, we can assume w.l.o.g. that $\gamma > \epsilon/2$, for otherwise we can neglect the variable x_i (and hence the perceptron g) without introducing much error. To see this, let f' be the function obtained from f by ignoring the perceptron g. It is easy to see that f may differ from f' only if $\mathbf{x}$ is such that

$$f^{(s)} = a^{(s)} \qquad \text{for all } s$$

That is,

$$P(f \neq f') \leq \prod_s P(f^{(s)} = a^{(s)})$$

Under these assumptions, the gap is wide enough to be estimated efficiently using a sample of polynomial size (lemma 2).

Once we determine the weight values, we reduce f to a monotone μ-PDL by changing x_i to $1 - x_i$ whenever $w_i = -1$. In the following, we concentrate on the monotone case.

The next step is to infer the architecture of the network. It turns out that the effect measure contains enough information to determine which variables appear at a given level, and among those, which are siblings. First, some facts about the effects.

Lemma 8 *Let f be a monotone μ-PDL. Let x_i and x_j be two influential variables in f. Assume that $x_i \in g$. Then*

1. *If x_i and x_j are siblings:*

$$\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}_I = \frac{P(g=1|x_i = x_j = 1) - P(g=1|x_i = 1, x_j = 0)}{P(g=1|x_j = 1) - P(g=1|x_j = 0)}.$$

2. *If x_i and x_j are not siblings, x_i is not deeper than x_j, and lca(x_i,x_j) is an OR:*

$$\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}_{II} = \frac{1 - P(g=1|x_i = 1)}{1 - P(g=1)}.$$

3. *If x_i and x_j are not siblings, x_i is not deeper than x_j, and lca(x_i,x_j) is an AND:*

$$\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}_{III} = \frac{P(g=1|x_i = 1)}{P(g=1)}.$$

Moreover, if g contains p variables and has a renormalized threshold v:

$$\mathrm{Eff}_I = 2\frac{v-1}{p-1} \quad ; \quad \mathrm{Eff}_{II} = 1 - \frac{\binom{p-1}{v-1}}{\sum_{i=0}^{v-1}\binom{p}{i}} \quad ; \quad \mathrm{Eff}_{III} = 1 + \frac{\binom{p-1}{v-1}}{\sum_{i=v}^{p}\binom{p}{i}} \tag{17}$$

$$\mathrm{Eff}_I - \mathrm{Eff}_{II} \geq \frac{1}{p^2} \geq \frac{1}{n^2} \qquad \text{for} \quad v \geq 2 \tag{18}$$

$$\mathrm{Eff}_I - \mathrm{Eff}_{III} \leq -\frac{1}{2p} \leq -\frac{1}{2n} \qquad \text{for} \quad v \leq p-1 \tag{19}$$

Proof: The three different expressions Eff_I, Eff_{II}, and Eff_{III} can be easily derived by induction on the depth of f. The derivation of eq. 17 is straightforward, using simple counting arguments under the uniform distribution. Eqs. 18 and 19 follow using the approximation introduced in lemma 1 (see the proof of lemma 5).□

Again, note that the effect gaps established in this lemma are wide enough to be estimated using a polynomial number of examples (lemma 2).

Let $X' \subseteq X$. We say that a variable $x_j \in X'$ is an *OR-potential-sibling* of $x_i \in X'$ (with respect to X') if $\text{Eff}(x_i, x_j) \geq \text{Eff}(x_i, x_k)$ for all $x_k \in X'$.

Likewise, we say that a variable $x_j \in X'$ is an *AND-potential-sibling* of $x_i \in X'$ (w.r.t. X') if $\text{Eff}(x_i, x_j) \leq \text{Eff}(x_i, x_k)$ for all $x_k \in X'$. The following lemma enables us to determine which variables appear in a given level, and among these, which are siblings.

Lemma 9 *Let f be a monotone μ-PDL and let $X' \subseteq X$. Assume that the deepest gate Γ fed by all variables in X' is an OR. Let $x_i \in X'$. Then, x_i is not a zero-level variable with respect to Γ iff either*

1. *there exist variables x_j, x_k, x_l such that: $\text{Eff}(x_i, x_j) \neq \text{Eff}(x_i, x_k) \neq \text{Eff}(x_i, x_l)$, or*
2. *there exists x_k such that: x_k is an an OR-potential-sibling of x_i and x_k is not a zero-level variable, or*
3. *there exist variables x_j, $x_k \in X'$ such that, for some permutation $\{i_1, i_2, i_3\}$ of $\{i, j, k\}$:*
   x_{i_1} is an OR-potential-sibling of x_{i_3} (w.r.t. X'),
   x_{i_2} is an OR-potential-sibling of x_{i_3} (w.r.t. X'), but
   x_{i_1} is not an OR-potential-sibling of x_{i_2} (w.r.t. X').

Moreover, if x_i is a zero-level variable with respect to Γ, and x_j is OR-potential-sibling of x_i (w.r.t. X'), then x_i and x_j are siblings in f.
The same holds if we replace OR by AND in the lemma.

Proof: See Appendix A.□

Assume that the output gate Γ is an OR. First, we apply lemma 9 to determine the zero-level variables (w.r.t Γ). For each zero-level variable, say x_i, we determine the set of its siblings, call it T_i. There may be several different sets of zero-level siblings which will form the various perceptrons that feed *immediately* the output gate Γ. A similar process applies if the output gate is an AND, using the AND-potential-siblings technique in lemma 9. We repeat this recursively, removing from X the variables already used and noting that the gates switch from OR to AND (or vice versa). [3]

The last step of the algorithm is to estimate the threshold of each perceptron g. Two cases are possible:

case 1: g is a single-variable perceptron. Then the only possible threshold value is 1.

case 2: g is a multi-variable perceptron. Let p be the number of variables in g and let v be its renormalized threshold. Let x_i, $x_j \in g$. Then (eq. 17):

$$\text{Eff}(x_i, x_j) = \text{Eff}_I = 2\frac{v-1}{p-1}$$

[3] In practice, one has to try both AND and OR as output gate and then choose the best solution.

Thus, a good estimation of $\mathrm{Eff}(x_i, x_j)$ yields a good estimation of the renormalized threshold v (and hence the original threshold).

By a standard Chernoff bounds analysis (lemma 2), one can show that a sample polynomial in $(n,1/\epsilon,1/\delta)$ is sufficient to ensure that, with high probability, the different influences and effects are estimated to within a sufficient precision. The number m of examples needed is $O((n^8/\epsilon^4)\ln(n^2/\delta))$, the same as for μ-PUs.

Finally, it takes $O(m)$ units of time to estimate a conditional probability using a sample of size m. Thus, it takes $O(m \times n^2)$ units of times to estimate the influences and the effects. Further, for each level of the μ-PDL, it takes $O(n)$ units of time to determine the OR(AND)-potential-siblings of each variable x_i, and it takes $O(n^3)$ units of time to determine whether a variable x_i appears in that level (at most, one has to check each pair/triplet of variables). Since there are n variables and at most n levels in the μ-PDL, the time necessary to infer the architecture is $O(n^5)$. Hence, the overall running time of the algorithm is $O(m \times n^2 + cn^5)$, where c is a constant independent of (n, ϵ, δ).

Theorem 10 *The class of μ-PDL with binary weights is PAC learnable under the uniform distribution.*

4.4 Learning Generalized μ-Perceptron Decision Lists

We extend the results of the previous section to handle the case where the target function f is a generalized μ-PDL (see fig. 4).

We assume w.l.o.g. that the target network is such that all gates belonging to a given level are of the same type (*i.e.* either all ANDs or all ORs) and that levels of type AND alternate with levels of type OR since two consecutive gates of the same type can always be merged. We also assume w.l.o.g. that the target network has the maximum possible number of perceptrons.

First, we note that lemmas 7, 8, and 9 hold also for generalized μ-PDLs. In fact, the problem of learning generalized μ-PDLs presents only one difficulty that is not encountered in learning μ-PDLs: because a given level in the network may contain more than one gate, we need to determine not only which variables appear at that level and which are siblings, but also which variables appear in the *same* subtree rooted at one of the gates of that level. This extra difficulty can be solved fairly easily. The following subroutine, when called with a set X' and a variable x_i, returns the set $S \subseteq X'$ of all variables that feed some AND gate fed by x_i:

AND-test:

1. Set $S = \{x_i\}$.
2. For each variable $x_j \in X'$ and $x_j \notin S$:
   If there exists $x_k \in S$ such that $\mathrm{Eff}(x_k, x_j) + \mathrm{Eff}(x_j, x_k) > 2$, set $S = S \cup \{x_j\}$ and go to 2.
3. If no variable can be added, return S.

Likewise, the following subroutine, when called with a set X' and a variable x_i, returns the set $S \subseteq X'$ of all variables that feed some OR gate fed by x_i:

OR-test:

1. Set $S = \{x_i\}$.
2. For each variable $x_j \in X'$ and $x_j \notin S$:
   If there exists $x_k \in S$ such that $\mathrm{Eff}(x_k, x_j) + \mathrm{Eff}(x_j, x_k) < 2$, set $S = S \cup \{x_j\}$ and go to 2.
3. If no variable can be added, return S.

The intuition behind the above subroutines is that if two variables meet at an AND gate, setting one of them to 1 increases the influence of the other, whereas if they meet at an OR gate, setting one of them to 1 decreases the influence of the other. The following lemma strengthens this intuition by showing that there is a measurable gap between the two cases.

Lemma 11 *Let f be a monotone generalized μ-PDL. Let x_i and x_j be two variables in f. Then, if x_i and x_j are not siblings*

$$\mathrm{Eff}(x_i, x_j) + \mathrm{Eff}(x_j, x_i) > 2 + \frac{1}{2}\left(\mathrm{Infl}(x_i) + \mathrm{Infl}(x_j)\right) \qquad \text{if} \quad lca(x_i, x_j) \ \text{ is an AND}$$

$$\mathrm{Eff}(x_i, x_j) + \mathrm{Eff}(x_j, x_i) < 2 - \frac{1}{2}\left(\mathrm{Infl}(x_i) + \mathrm{Infl}(x_j)\right) \qquad \text{if} \quad lca(x_i, x_j) \ \text{ is an OR}$$

Proof: We prove the case where the $lca(x_i, x_j)$ Γ is an AND. Let $f^{(1)}$ be the subformula computed by gate Γ. Then $f^{(1)}$ can be decomposed as

$$f^{(1)} = f^{(2)} \wedge f^{(3)}$$

where the subformula $f^{(2)}$ ($f^{(3)}$) depends only on x_i (x_j). Then,

$$\begin{aligned}
\mathrm{Eff}(x_i, x_j) = \mathrm{Eff}_{III} &= \frac{P(f^{(2)} = 1 | x_i = 1)}{P(f^{(2)} = 1)} \\
&= 1 + \frac{P(f^{(2)} = 1 | x_i = 1) - P(f^{(2)} = 1)}{P(f^{(2)} = 1)} \\
&> 1 + [\ P(f^{(2)} = 1 | x_i = 1) - P(f^{(2)} = 1)\ ] \\
&> 1 + \frac{1}{2}\mathrm{Infl}(x_i) \qquad (20)
\end{aligned}$$

under the uniform distribution. Moreover:

$$\begin{aligned}
\mathrm{Eff}(x_j, x_i) &= \frac{P(f^{(3)} = 1 | x_j = 1)}{P(f^{(3)} = 1)} \\
&> 1 + \frac{1}{2}\mathrm{Infl}(x_j) \qquad (21)
\end{aligned}$$

Combining eqs. 20 and 21 yields the desired result. A similar argument applies to the case where the $lca(x_i, x_j)$ Γ is an OR.□

Assume that the output gate Γ is an OR (a similar procedure applies when Γ is an AND). The zero-level variables w.r.t. Γ are determined as in the previous section, using the OR-potential-siblings technique. These variables are then removed and the AND-test is invoked to determine which set of variables feed the *same* AND gate in the next level. There may be several different sets determined in this manner; each set S_i will be assigned an AND gate Γ_i in that level. Then, for each set S_i we use the AND-potential-siblings technique to determine which variables are zero-level w.r.t. Γ_i. This may yield several different subsets, which will form the various perceptrons that feed *immediately* the gate Γ_i. We repeat this subdivision process recursively, removing the variables already used and noting that the gates switch from OR to AND (or vice versa). [4]

Finally, the threshold of each perceptron can be estimated by the same method used in the previous section.

By a standard Chernoff bounds analysis, one can show that a sample polynomial in $(n,1/\epsilon,1/\delta)$ is sufficient to ensure that, with high probability, the different influences and effects are estimated to within a sufficient precision. The number m of examples needed is $O((n^8/\epsilon^4)\ln(n^2/\delta))$. The overall running time of the algorithm is, as for μ-PDLs, $O(m\times n^2+n^5)$. (The extra cost, with respect to μ-PDLs, of running AND(OR)-tests is $O(n^4)$: each AND(OR)-test call costs $O(n^2)$ and the number of calls to AND(OR)-test at each level is $O(n)$.)

Theorem 12 *The class of generalized μ-PDL with binary weights is PAC learnable under the uniform distribution.*

5 Conclusion and Open Problems

We have presented statistical methods for PAC learning different classes of nonoverlapping perceptron networks with binary weights on the uniform distribution. These algorithms work by estimating statistical quantities that yield enough information to infer, with high probability, the target network. Because of their statistical nature, these algorithms are robust against a large amount of random classification noise (Kearns, 1993).

The hardness results of (Blum & Rivest, 1988; Judd, 1988) suggest that one can not avoid the training difficulties simply by considering only very simple neural networks. The results of this paper suggest that the combination of simple networks and reasonable distributions can overcome the training difficulties and yield successful learning algorithms (see also Marchand and Golea, 1993).

The general class of nonoverlapping perceptron networks has recently been shown to be PAC learnable from examples and membership queries (Hancock et al., 1994). It is still an open problem whether this class is PAC learnable from examples only on the uniform distribution. A more tractable problem would be to extend the techniques of this paper to handle the class of nonoverlapping perceptron networks with binary weights and arbitrary thresholds. Another interesting direction would be to extend the results to more general classes of distributions like product distributions. We are currently investigating these possibilities.

[4] Again, in practice one has to try both AND and OR as output gate and then choose the best solution.

References

Bahadur R. (1960). Some Approximations to the Binomial Distribution Function. *Annals Math. Stat.*, Vol. 31, p. 43.

Barkai E., Hansel D., and Kanter I. (1990). Statistical mechanics of multilayer neural networks. *Physical Review Letters*, Vol. 65, p. 2312.

Blum A. & Rivest R.L. (1988). Training a 3-node neural network is NP-complete'. In *Proc. of the 1st Workshop on Computational Learning Theory*, p. 9. Morgan Kaufmann.

Bshouty N.H., Hancock T.R., and Hellerstein L. (1992). Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, p. 1. New York: ACM Press.

Golea M. (1994). Average Case Analysis of a Learning Algorithm for μDNF Expressions. *submitted*.

Golea M., Marchand M., and Hancock T.R. (1992). On Learning μ-Perceptron Networks with Binary Weights. In Hanson S., Cowan J., and Giles C. (Eds), *Advances in Neural Information Processing Systems*, Vol. 5, p. 591. San Mateo, CA: Morgan Kaufmann.

Goldman S., Kearns M., and Schapire R. (1990). Exact identification of circuits using fixed points of amplification functions. In *Proceedings of the 31st Symposium on Foundations of Computer Science*.

T.H. Cormen, C.E. Leiserson, and R.L. Rivest (1990). Introduction to Algorithms. M.I.T. Press.

Hancock T., Golea M., and Marchand M. (1994). Learning Nonoverlapping Perceptron Networks From Examples and Membership Queries. *Machine Learning*, vol. 16, p. 161.

Haussler D. (1990). Probably Approximately Correct Learning. Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90). p. 1101.

Judd S. (1988). On the complexity of loading shallow neural networks. *Journal of Complexity*, Vol. 4, p. 177.

Karmarkar N. (1984). A New Polynomial Time algorithm for Linear Programming. *Combinatorica*, Vol. 4, p. 373.

Kharitonov, M. (1993). Cryptographic hardness of distribution specific learning. *Proceedings of the Twenty Fifth Annual ACM Symposium on the Theory of Computation*, p. 372.

Kearns M. (1993). Efficient Noise-Tolerant Learning from Statistical Queries. *Proceedings of the Twenty Fifth Annual ACM Symposium on the Theory of Computation*, p. 392.

Kearns M., Li M., Pitt L., and Valiant L. (1987). On the learnability of boolean formulas. *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, p. 285. New York: ACM Press.

Kearns M. & Valiant L. (1989). Cryptographic limitations on learning boolean formulae and finite automata. *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, p. 433. New York: ACM Press.

Lin J.H. & Vitter J.S. (1991). Complexity results on learning by neural nets. *Machine Learning*, Vol. 6, p. 211.

Marchand M. & Golea M. (1993). On Learning Simple Neural Concepts: from Halfspace Intersections to Neural Decision Lists. *Network: Computation in Neural Systems*, Vol. 4, p. 67.

Pagallo G. & Haussler D. (1989). A greedy method for learning μDNF functions under the uniform distribution. Technical Report UCSC-CRL-89-12, Santa Cruz: Dept. of Computer and Information Science, University of California at Santa Cruz.

Pitt L. & Valiant L.G. (1988). Computational Limitations on Learning from Examples. *J. ACM*, Vol. 35, p. 965.

Rivest R.L. (1987). Learning Decision Lists. *Machine Learning*, Vol. 2, p. 229.

Schapire R.E. (1991). Learning probabilistic read-once formulas on product distributions. *Proc. of the 4th Annual Workshop on Computational Learning Theory*, p. 184. San Mateo, CA: Morgan Kaufmann.

Valiant L.G. (1984). A theory of the learnable. *Communications of the ACM*, Vol. 27, p. 1134.

Watkin T., Rau A., and Biehl M. (1993). The Statistical Mechanics of Learning a Rule. *Rev. Mod. Phys.*, Vol. 65, p. 499.

Appendix A: Proof of lemma 9

Assume that Γ is an OR (the case where Γ is an AND can be treated similarly). It is clear that if $x_i \in g^{(a)}$ is a zero-level variable w.r.t. Γ then (lemma 8)

$$\mathrm{Eff}(x_i, x_j) = \begin{cases} \mathrm{Eff}_I & \text{if } x_j \in g^{(a)} \\ \mathrm{Eff}_{II} & \text{if } x_j \notin g^{(a)} \end{cases}$$

Further, $\mathrm{Eff}_I > \mathrm{Eff}_{II}$. Thus, the OR-potential-siblings of a zero-level variable x_i are exactly its siblings. This implies that, if x_j is an OR-potential-sibling of x_i, x_k is an OR-potential-sibling of x_j, then x_k is also an OR-potential-sibling of x_i. This proves the "only if" part of the lemma.

To prove the "if" part, we consider the different situations and apply the facts of lemma 8:

Case 1: $x_i \in g^{(a)}$ is an immediate input to an OR gate. Then, there exist x_j, x_k, and x_r as shown in fig. 6a. Referring to this figure, we can write:

$$\mathrm{Eff}(x_i, x_j) = \frac{P(f_2 = 1 | x_i = 1)}{P(f_2 = 1)} \geq 1 + \frac{1}{2}\mathrm{Infl}(x_i)$$

$$\mathrm{Eff}(x_i, x_k) = \frac{1 - P(g^{(a)} = 1 | x_i = 1)}{1 - P(g^{(a)} = 1)} \leq 1 - \frac{1}{2}\mathrm{Infl}(x_i)$$

For any other variable, say x', either
1) $\mathrm{Eff}(x_i, x_j) \neq \mathrm{Eff}(x_i, x') \neq \mathrm{Eff}(x_i, x_k)$. In this case x_i is not a zero-level variable by condition 1 of the lemma and we are done, or
2) $\mathrm{Eff}(x_i, x') = \mathrm{Eff}(x_i, x_j)$ or $\mathrm{Eff}(x_i, x') = \mathrm{Eff}(x_i, x_k)$. In either case,

$$x_j \quad \text{is an OR} - \text{Potential} - \text{sibling of } x_i \tag{22}$$
$$x_k \quad \text{is not an OR} - \text{Potential} - \text{sibling of } x_i \tag{23}$$

because $\mathrm{Eff}(x_i, x_j) > \mathrm{Eff}(x_i, x_k)$.

We can also write:

$$\mathrm{Eff}(x_j, x_i) = \mathrm{Eff}(x_j, x_k) = \frac{P(g^{(b)} = 1 | x_j = 1)}{P(g^{(b)} = 1)} \geq 1 + \frac{1}{2}\mathrm{Infl}(x_j)$$

$$\mathrm{Eff}(x_j, x_r) = \frac{1 - P(f_1 = 1 | x_j = 1)}{1 - P(f_1 = 1)} \leq 1 - \frac{1}{2}\mathrm{Infl}(x_j)$$

Again, for any other variable, say x', either
1) $\mathrm{Eff}(x_j, x_i) \neq \mathrm{Eff}(x_j, x') \neq \mathrm{Eff}(x_j, x_r)$. In this case x_j is not a zero-level variable by condition 1 of the lemma, and hence x_i is not a zero-level variable (by 22 and condition 2 of the lemma), or
2) $\mathrm{Eff}(x_j, x') = \mathrm{Eff}(x_j, x_i)$ or $\mathrm{Eff}(x_j, x') = \mathrm{Eff}(x_i, x_r)$. In either case,

$$x_i \text{ and } x_k \quad \text{are OR} - \text{Potential} - \text{siblings of } x_j \tag{24}$$

Then x_i is not a zero-level variable by 23, 24, and condition 3 of the lemma.

Case 2: $x_i \in g^{(a)}$ is an immediate input to an AND gate. Two subcases are possible.

Case 2.1: x_i's parent $g^{(a)}$ is a multi-variable perceptron. Then there exist x_j, x_r, x_k as shown in figure 6b (the existence of x_k is guaranteed by the fact that the AND gate has at least two inputs). Referring to this figure, we can write:

$$\text{Eff}(x_i, x_j) = \text{Eff}_I = \frac{P(g^{(a)} = 1 | x_i = x_j = 1) - P(g^{(a)} = 1 | x_i = 1, x_j = 0)}{P(g^{(a)} = 1 | x_j = 1) - P(g^{(a)} = 1 | x_j = 0)}$$

$$\text{Eff}(x_i, x_k) = \text{Eff}_{III} = \frac{P(g^{(a)} = 1 | x_j = 1)}{P(g^{(a)} = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_i)$$

$$\text{Eff}(x_i, x_r) = \frac{1 - P(f_1 = 1 | x_i = 1)}{1 - P(f_1 = 1)} \leq 1 - \frac{1}{2}\text{Infl}(x_i)$$

Thus, $\text{Eff}(x_i, x_k) > \text{Eff}(x_i, x_j), \text{Eff}(x_i, x_r)$.
For any other variable, say x', either
1) $\text{Eff}(x_i, x_k) \neq \text{Eff}(x_i, x') \neq \text{Eff}(x_i, x_j), \text{Eff}(x_i, x_r)$. In this case x_i is not a zero-level variable by condition 1 of the lemma, or
2) $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_k)$ or $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_j) = \text{Eff}(x_i, x_r)$. In either case,

$$x_k \quad \text{is an OR} - \text{Potential} - \text{sibling of } x_i \tag{25}$$

$$x_j \quad \text{is not an OR} - \text{Potential} - \text{sibling of } x_i \tag{26}$$

We can also write:

$$\text{Eff}(x_k, x_i) = \text{Eff}(x_k, x_j) = \frac{P(f_2 = 1 | x_k = 1)}{P(f_2 = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_k)$$

$$\text{Eff}(x_k, x_r) = \frac{1 - P(f_1 = 1 | x_k = 1)}{1 - P(f_1 = 1)} \leq 1 - \frac{1}{2}\text{Infl}(x_k)$$

Thus, $\text{Eff}(x_k, x_i) = \text{Eff}(x_k, x_j) > \text{Eff}(x_k, x_r)$. Again, for any other variable, say x', either
1) $\text{Eff}(x_k, x_i) \neq \text{Eff}(x_k, x') \neq \text{Eff}(x_k, x_r)$. In this case x_k is not a zero-level variable by condition 1 of the lemma, and hence x_i is not a zero-level variable by 25 and condition 2 of the lemma, or
2) $\text{Eff}(x_k, x') = \text{Eff}(x_k, x_i)$ or $\text{Eff}(x_k, x') = \text{Eff}(x_i, x_r)$. In either case,

$$x_i \text{ and } x_j \quad \text{are} \quad \text{OR} - \text{Potential} - \text{siblings of } x_k \tag{27}$$

Then x_i is not a zero-level variable by 26, 27, and condition 3 of the lemma.

Case 2.2: x_i's parent $g^{(a)}$ is a single-variable perceptron. Two subcases are possible: either there exists an OR gate (fig. 6c) or a multivariable perceptron $g^{(b)}$ (fig. 6d) at the same depth as $g^{(a)}$. This is true because if neither exist, then the AND gate fed immediately by $g^{(a)}$ can be considered as a perceptron that feeds an OR gate in the previous level.

Case 2.2.1: Referring to fig. 6c, we can write:

$$\text{Eff}(x_i, x_j) = \text{Eff}(x_i, x_k) = \frac{P(g^{(a)} = 1 | x_i = 1)}{P(g^{(a)} = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_i)$$

$$\text{Eff}(x_i, x_r) = \frac{1 - P(f_1 = 1 | x_i = 1)}{1 - P(f_1 = 1)} \leq 1 - \frac{1}{2}\text{Infl}(x_i)$$

Thus, $\text{Eff}(x_i, x_j) = \text{Eff}(x_i, x_k) > \text{Eff}(x_i, x_r)$. For any other variable, say x', either
1) $\text{Eff}(x_i, x_k) \neq \text{Eff}(x_i, x') \neq \text{Eff}(x_i, x_r)$. In this case x_i is not a zero-level variable by condition 1 of the lemma, or
2) $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_k)$ or $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_r)$. In either case,

$$x_k \text{ and } x_j \quad \text{are OR} - \text{Potential} - \text{siblings of } x_i \tag{28}$$

We can also write:

$$\text{Eff}(x_j, x_i) = \frac{P(f_2 = 1 | x_j = 1)}{P(f_2 = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_j)$$

$$\text{Eff}(x_j, x_k) = \frac{1 - P(f_3 = 1 | x_j = 1)}{1 - P(f_3 = 1)} \leq 1 - \frac{1}{2}\text{Infl}(x_j)$$

Thus, $\text{Eff}(x_j, x_i) > \text{Eff}(x_j, x_k)$. Hence

$$x_k \quad \text{is not an OR} - \text{Potential} - \text{sibling of } x_j \tag{29}$$

Combining 28 and 29 yields the desired result.

Case 2.2.2: Referring to fig. 6d, we can write:

$$\text{Eff}(x_i, x_j) = \text{Eff}(x_i, x_k) = \frac{P(g^{(a)} = 1 | x_i = 1)}{P(g^{(a)} = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_i)$$

$$\text{Eff}(x_i, x_r) = \frac{1 - P(f_1 = 1 | x_i = 1)}{1 - P(f_1 = 1)} \leq 1 - \frac{1}{2}\text{Infl}(x_i)$$

Thus, $\text{Eff}(x_i, x_j) = \text{Eff}(x_i, x_k) > \text{Eff}(x_i, x_r)$. For any other variable, say x', either
1) $\text{Eff}(x_i, x_k) \neq \text{Eff}(x_i, x') \neq \text{Eff}(x_i, x_r)$. In this case x_i is not a zero-level variable by condition 1 of the lemma, or
2) $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_k)$ or $\text{Eff}(x_i, x') = \text{Eff}(x_i, x_r)$. In either case,

$$x_k \text{ and } x_j \quad \text{are OR} - \text{Potential} - \text{siblings of } x_i \tag{30}$$

We can also write:

$$\text{Eff}(x_j, x_i) = \text{Eff}_{III} = \frac{P(g^{(b)} = 1 | x_j = 1)}{P(g^{(b)} = 1)} \geq 1 + \frac{1}{2}\text{Infl}(x_j)$$

$$\text{Eff}(x_j, x_k) = \text{Eff}_I = \frac{P(g^{(b)} = 1 | x_j = x_k = 1) - P(g^{(b)} = 1 | x_j = 1, x_k = 0)}{P(g^{(b)} = 1 | x_j = 1) - P(g^{(b)} = 1 | x_k = 0)}$$

Thus, $\text{Eff}(x_j, x_i) > \text{Eff}(x_j, x_k)$. Hence

$$x_k \quad \text{is not an OR} - \text{Potential} - \text{sibling of } x_j \tag{31}$$

Combining 30, and 31 yields the desired result. □

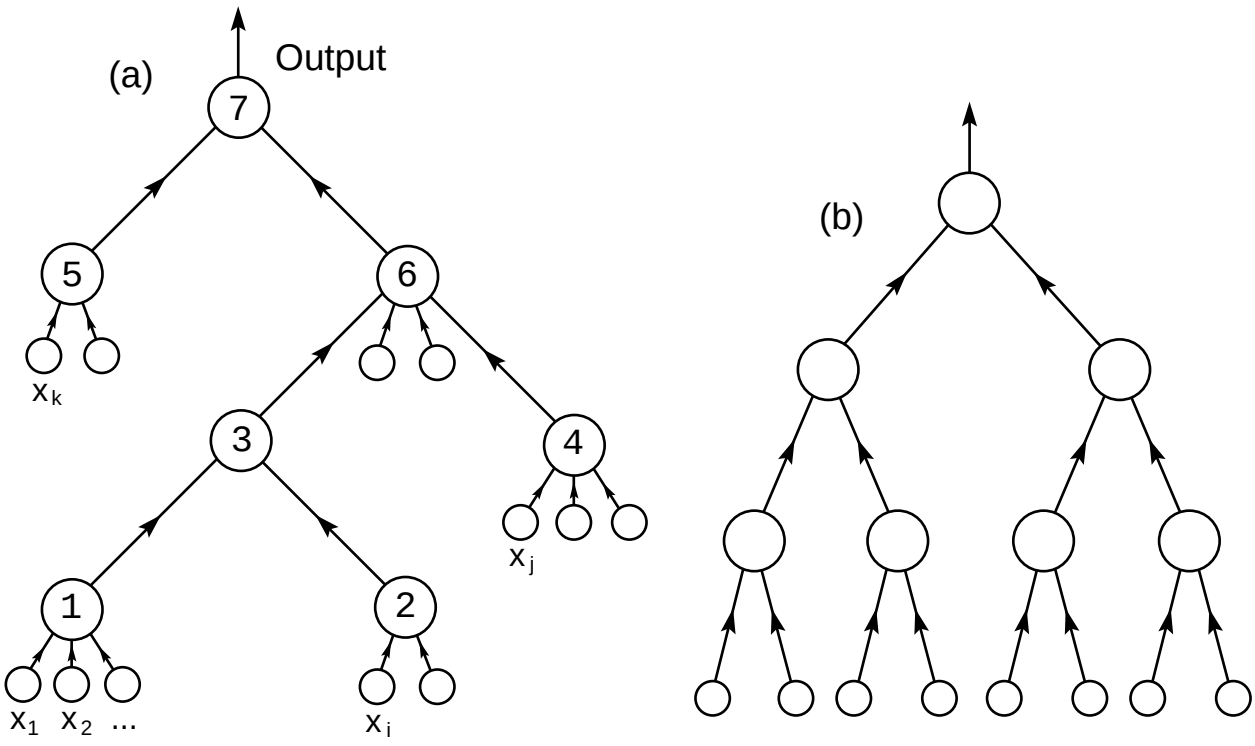

Figure 1:

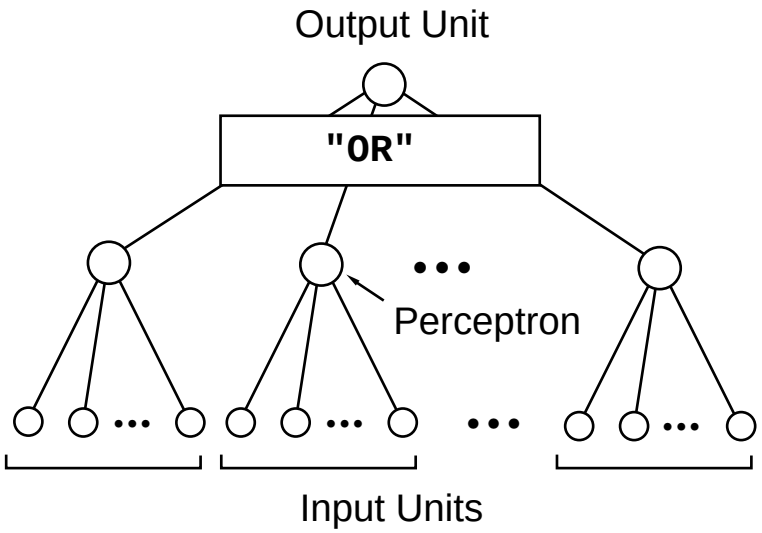

Figure 2:

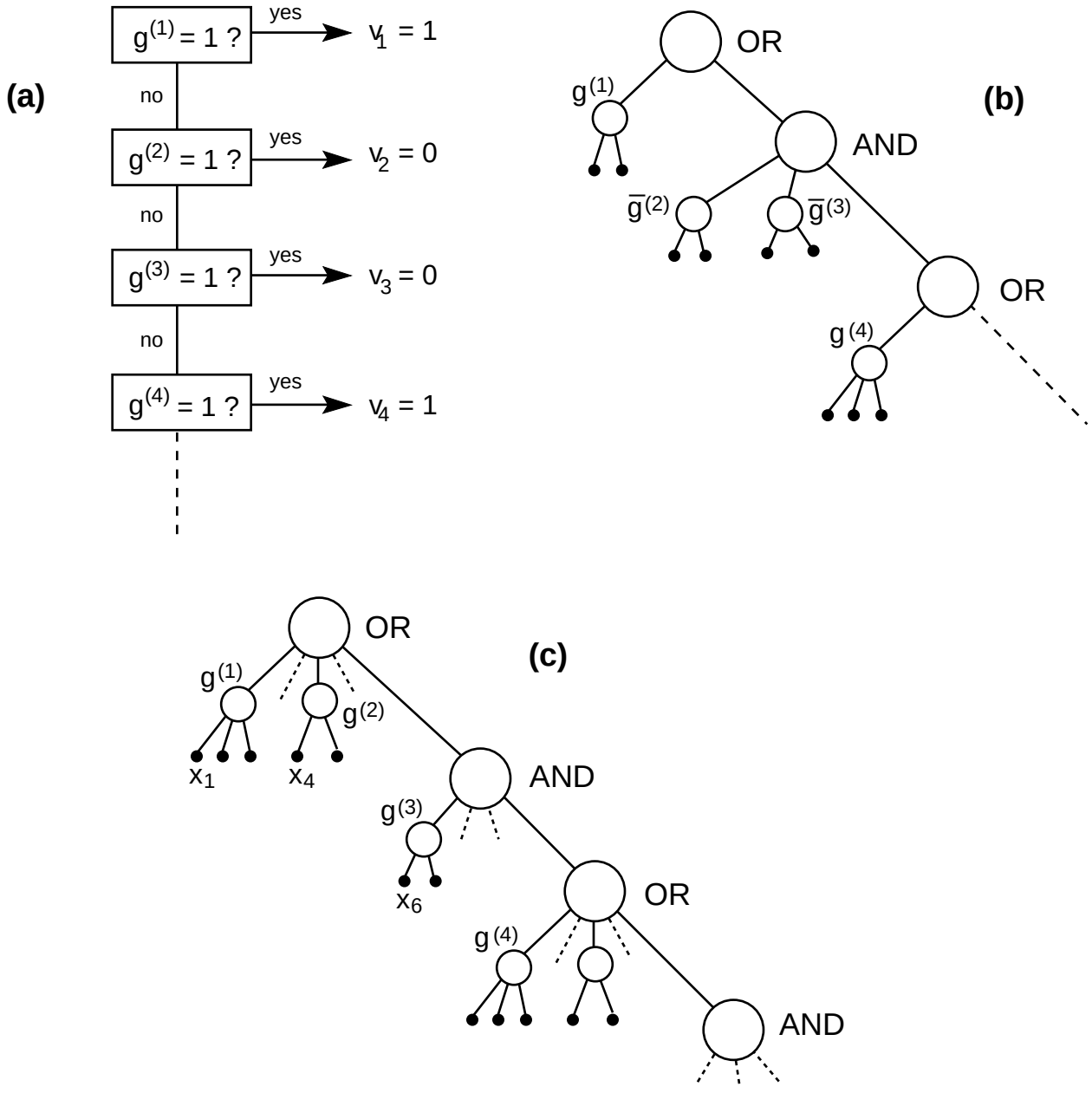
(a)
g(1) = 1 ?
yes
v1 = 1
no
g(2) = 1 ?
yes
v2 = 0
no
g(3) = 1 ?
yes
v3 = 0
no
g(4) = 1 ?
yes
v4 = 1
(b)
OR
g(1)
AND
g(2)
g(3)
OR
g(4)
(c)
OR
g(1)
g(2)
x1
x4
AND
g(3)
x6
OR
g(4)
AND

Figure 3:

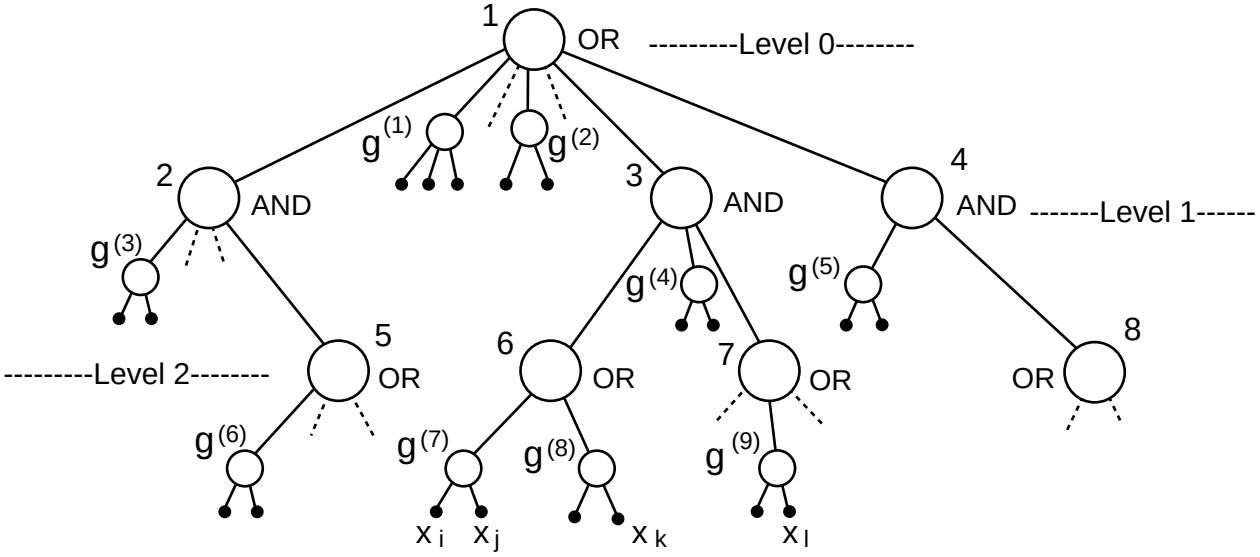
1
OR
---------Level 0--------
g(1)
g(2)
2
AND
3
AND
4
AND
-------Level 1------
g(3)
g(4)
g(5)
5
OR
6
OR
7
OR
8
OR
---------Level 2--------
g(6)
g(7)
g(8)
g(9)
x i
x j
x k
x l

Figure 4:

Parameters: (n,ϵ,δ)
n is the number of input variables, ϵ is the accuracy parameter and δ is the confidence parameter.
Output: a μ-perceptron union h.
Description:

1. Call $m = (80)^4 \left(\frac{n^6(n+2)^2}{\epsilon^4}\right) \ln \frac{16n^2}{\delta}$ examples. This sample will be used to estimate the different probabilities.
   Initialize the hypothesis h to the constant function 1.
2. (Are most examples positive?) If $\hat{P}(f = +1) \geq (1 - \frac{3\epsilon}{4})$ then return h.
3. (Are most examples negative?) If $\hat{P}(f = +1) \leq \frac{3\epsilon}{4}$ then return $h = 0$.
4. Set $\gamma = \frac{\epsilon}{2}$ and $\rho = \frac{\epsilon}{2n}$.
5. (Determine the weight values) For each input variable x_i:
   (a) If $\hat{\text{Infl}}(x_i) > \frac{1}{2}\frac{\gamma\rho}{1(n+2)}$, set $w_i = 1$.
   (b) Else if $\hat{\text{Infl}}(x_i) < -\frac{1}{2}\frac{\gamma\rho}{2(n+2)}$, set $w_i = -1$ and change x_i to $1 - x_i$.
   (c) Else set $w_i = 0$ (x_i is not an influential variable and is thus neglected in what follows).
6. (Build μ-perceptron union)
   Let $X = \{x_1, x_2..., x_n\}$. Initialize $X' = X$ and $a = 1$.
   (a) If $X' = \emptyset$, return h.
   (b) Choose $x_i \in X'$. Set $X' = X' - \{x_i\}$. Set $a = a + 1$.
   (c) Divide the set $X - \{x_i\}$ into two (disjoint) sets S_i and $(X - \{x_i\} - S_i)$ such that, for all $x_j \in S_i$ and all $x_k \in (X - \{x_i\} - S_i)$

$$\hat{\text{Eff}}(x_i, x_j) - \hat{\text{Eff}}(x_i, x_k) > \frac{1}{2n^2}$$

   Then S_i is the set of x_i's siblings.
   (d) The set of variables that appear in $g^{(a)}$ is $S_i \cup x_i$. Let $p = |S_i \cup x_i|$.
   (e) Set v, the normalized threshold of $g^{(a)}$, such that: $\left[\hat{C}(x_i, x_j) \times (p-1)\right] = 2(v-1)$, where $[q]$ denotes the closest integer to q.
   (f) Set $X' = X' - S_i$. Set $h = h \vee g^{(a)}$. Go to step 6a.

Figure 5:

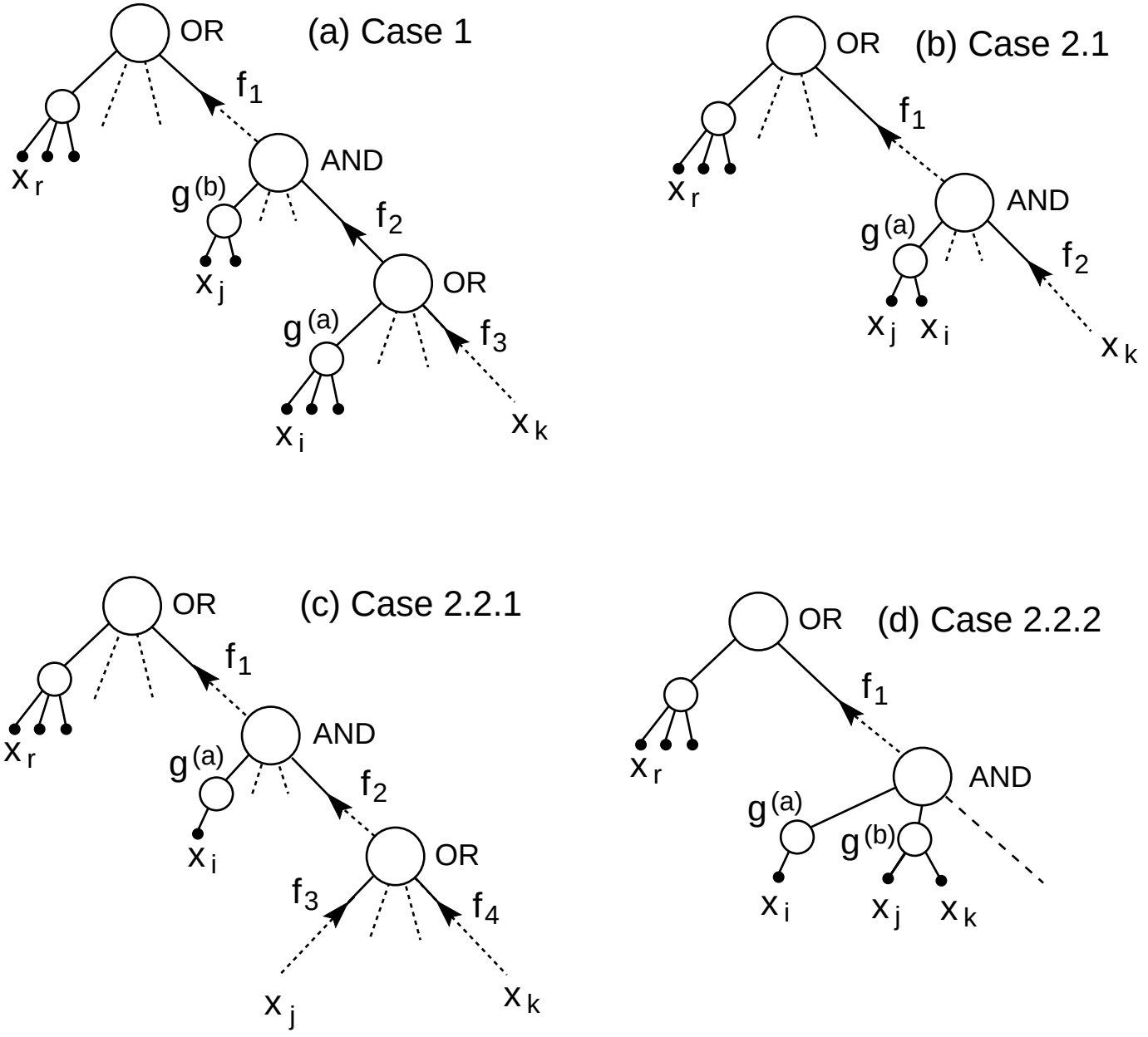
OR
(a) Case 1
f1
AND
xr
g(b)
f2
xj
OR
g(a)
f3
xi
xk
OR
(b) Case 2.1
f1
xr
AND
g(a)
f2
xj
xi
xk
OR
(c) Case 2.2.1
f1
xr
AND
g(a)
f2
xi
OR
f3
f4
xj
xk
OR
(d) Case 2.2.2
f1
xr
AND
g(a)
g(b)
xi
xj
xk

Figure 6:

Figure Captions

Fig. 1 (a) Architecture of a nonoverlapping (μ) perceptron network. Small circles represent input variables and big circles represent (computing) perceptrons. Note that each node, including the input variables but excluding the output, has only one outgoing connection. (b) a layered nonoverlapping (μ) perceptron network.

Fig. 2 A two-layer network representing a μ-perceptron union. Note that each input unit is connected to one and only one perceptron (hidden unit). The output unit computes an OR function.

Fig. 3 (a) Example of a perceptron decision list. (b) the equivalent feedforward network: $\overline{g}$ denotes the negation of g, i.e., $\overline{g}(\mathbf{x}) = 1 - g(\mathbf{x})$. Note that if g is a perceptron with binary weights, so is $\overline{g}$. Thus, negated perceptrons can be replaced by equivalent, non-negated perceptrons. (c) A network representing a general perceptron decision list.

Fig. 4 A network representing a generalized perceptron decision list. As an example of the definitions of section 2: x_i and x_j are siblings. x_i and x_k are not siblings. x_i's parent is perceptron $g^{(7)}$. The lca(x_i, x_l) is gate number 3. The gates that are fed by x_i are numbers 6, 3, and 1. The uncles of x_i are gates numbered 7, 2, and 4, as well as perceptrons $g^{(8)}, g^{(4)}, g^{(1)}$, and $g^{(2)}$. Perceptron $g^{(4)}$ and gate 7 have depth 1. The depth of x_l is 2. x_i is a zero-level variable w.r.t gate 6.

Fig. 5 An algorithm for learning μ-perceptron unions with binary weights on the uniform distribution.

Fig. 6 Proof of lemma 10. (a) case 1. (b) case 2.1. (c) case 2.2.1. (d) case 2.2.2.