Propositionalizing the EM algorithm by BDDs

Masakazu Ishihata[1], Yoshitaka Kameya[1], Taisuke Sato[1], and Shin-ichi Minato[2]

[1] Graduate School of Information Science and Engineering,
Tokyo institute of Technology
{ishihata,kameya,sato}@mi.cs.titech.ac.jp
[2] Graduate School of Information Science and Technology,
Hokkaido University
minato@ist.hokudai.ac.jp

Abstract. We propose an EM algorithm working on binary decision diagrams (BDDs). It opens a way to applying BDDs to statistical inference in general and machine learning in particular. We also present the complexity analysis of noisy-OR models.

1 Introduction

Binary decision diagrams (BDDs) have been popular as a basic tool for compactly representing boolean functions [1, 2]. In this paper[3] we present yet another application of BDDs. We propose a new EM algorithm that works on BDDs. Since the EM algorithm is a fundamental parameter learning algorithm for maximum likelihood estimation in statistics [4], our proposal opens a way to apply BDDs to statistical learning in general and to machine learning in particular.

2 The EM algorithm

We here describe our unsupervised learning setting and review the expectation-maximization (EM) algorithm [4]. First of all, we assume our problem domain is modeled with k boolean *random* variables $X_1, X_2, \ldots, X_k$, each taking 1 (true) and 0 (false) independently of each other. Let F be a boolean function composed of these k variables, and assume only the value of F is observable whereas those of the X_i's are not. Hereafter, to make notations simple, F is treated as a boolean random variable as well that takes the value of (the function) F. We then call F an *observable variable*, and the X_i's *basic variables*. The EM algorithm proposed in this paper aims to estimate the probabilities of basic variables being true from the observed values of F.

Let ϕ be an assignment of the set of basic variables $\mathbf{X} = \{X_1, X_2, \ldots, X_k\}$. ϕ maps each variable $X \in \mathbf{X}$ to its value $x \in \{0, 1\}$. We assume $\mathbf{X}$ is partitioned

[3] This is a shortened version of [3], which deals with zero-suppressed BDDs (ZBDDs) as well as BDDs.

into $\mathcal{S}$, sets of i.i.d. variables, and each partition $s \in \mathcal{S}$ has a *parameter* $\theta_{s,x}$, a common probability of $X \in s$ taking x. We use Φ to stand for the set of all assignments. Since the value $F = f \in \{0,1\}$ is uniquely determined by ϕ, F is a function $F(\phi) = f$ of assignments. Hence the set of assignments which make $F = f$ is written as $F^{-1}(f) = \{\phi \in \Phi \mid F(\phi) = f\}$. We introduce $\sigma_{s,x}(\phi) = |\{X \in s \mid \phi(X) = x\}|$ to denote the total number of i.i.d. variables in the partition s that takes a value x by ϕ. The EM algorithm we develop for the setting described above consists of two steps, the *expectation step* (E-step) and the *maximization step* (M-step), defined as follows:

- **E-step**: Compute the *conditional expectation* $E_{\boldsymbol{\theta}}[\sigma_{s,x}(\cdot) \mid F{=}f]$ by $\eta^x_{\boldsymbol{\theta}}[s]/P_{\boldsymbol{\theta}}(F{=}f)$, where:

$$\eta^x_{\boldsymbol{\theta}}[s] = \sum_{\phi \in F^{-1}(f)} \sigma_{s,x}(\phi) \prod_{s' \in \mathcal{S}} \prod_{x' \in \{1,0\}} (\theta_{s',x'})^{\sigma_{s',x'}(\phi)} \tag{1}$$

$$P_{\boldsymbol{\theta}}(F{=}f) = \sum_{\phi \in F^{-1}(f)} \prod_{s \in \mathcal{S}} \prod_{x \in \{1,0\}} (\theta_{s,x})^{\sigma_{s,x}(\phi)}. \tag{2}$$

- **M-step**: Update $\boldsymbol{\theta}$ to $\hat{\boldsymbol{\theta}}$ by $\hat{\theta}_{s,x} \propto E_{\boldsymbol{\theta}}[\sigma_{s,x}(\cdot) \mid F{=}f]$.

3 BDDs and the EM algorithm

A BDD is a rooted directed acyclic graph representing a boolean function as a disjunction of exclusive conjunctions. It has two terminal nodes, $\boxed{1}$ (true) and $\boxed{0}$ (false). Each nonterminal node n is labeled with a binary random variable denoted by $\mathit{Label}(n)$, and has two outgoing edges called 1-edge and 0-edge, indicating that $\mathit{Label}(n)$ takes 1 and 0, respectively. $\mathrm{Ch}^x(n)$ stands for a child node of n, connected by the x-edge ($x \in \{0,1\}$).

A *reduced ordered BDD (ROBDD)* is a BDD which is a unique representation of the target boolean function. Fig. 1 illustrates some different representations of a boolean function $F = (A \vee B) \wedge \bar{C}$. Fig. 1 (a) is a truth table, in which a row corresponds to an assignment ϕ for $\mathbf{X} = \{A, B, C\}$. One way to obtain the ROBDD for F (Fig. 1 (d)) is to consider a binary decision tree (BDT) (b) and apply two reduction rules, the deletion rule and the merging rule, as many times as possible to reach (d).

Consider a BDT like the one in Fig. 1 (b). In a BDT, there is a unique path π_ϕ from the root to a terminal for an assignment ϕ, in which every basic variable appears once as a node label. We rewrite Eq. 1 to Eq. 3 so that $\eta^x_{\boldsymbol{\theta}}[s]$ is computed on a BDD:

$$\eta^x_{\boldsymbol{\theta}}[s] = \sum_{\pi_\phi : \phi \in F^{-1}(f)} \sum_{n' \in \pi_\phi : L_{n'} \in s} \mathbf{1}_{\phi(L_{n'})=x} \prod_{n \in \pi_\phi} (\theta_{[L_n]})^{\phi(L_n)} (\theta_{\overline{[L_n]}})^{1-\phi(L_n)} \tag{3}$$

Here $n \in \pi_\phi$ says that the node n is on the path π_ϕ. L_n is a shorthand for $\mathit{Label}(n)$ and $[L_n]$ is the partition to which L_n belongs. $\mathbf{1}_{\phi(L_{n'})=x} = 1$ if $\phi(L_{n'}) = x$ is true, and 0 otherwise.

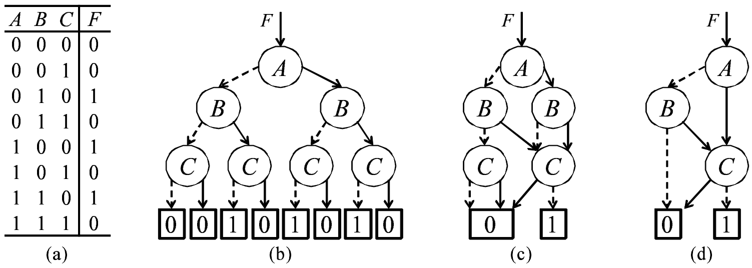

Fig. 1. Examples of (a) a truth table, (b) a binary decision tree (BDT), (c) a BDD which is ordered but is not reduced, (d) the ROBDD, for $F = (A \vee B) \wedge \bar{C}$.

4 The BDD-EM algorithm

We here present the BDD-EM algorithm which is an EM algorithm working on BDDs. There are four auxiliary procedures for the procedure BDD-EM(), i.e. IterateEM(), GetBackward(), GetForward() and GetExpectation().

Procedure: BDD-EM()
  Initialize all parameters $\boldsymbol{\theta}$;
  repeat
    IterateEM();
  until the parameters $\boldsymbol{\theta}$ converge;
end

Procedure: IterateEM()
  // E-step
  GetBackward();
  GetForward();
  GetExpectation();
  // M-step
  for each $s \in \mathcal{S}$ **do**
    $\theta_{s,1} \propto \eta^1_{\boldsymbol{\theta}}[s]/\mathcal{P}^f_{\boldsymbol{\theta}}[F]$;
    $\theta_{s,0} \propto \eta^0_{\boldsymbol{\theta}}[s]/\mathcal{P}^f_{\boldsymbol{\theta}}[F]$;
  end for
end

Procedure: GetBackward()
  $\mathcal{B}^1_{\boldsymbol{\theta}}[\boxed{1}] = 1$, $\mathcal{B}^1_{\boldsymbol{\theta}}[\boxed{0}] = 0$;
  $\mathcal{B}^0_{\boldsymbol{\theta}}[\boxed{1}] = 0$, $\mathcal{B}^0_{\boldsymbol{\theta}}[\boxed{0}] = 1$;
  $\mathcal{N} = \mathrm{Par}(\boxed{1}) \cup \mathrm{Par}(\boxed{0})$;
  // $\mathrm{Par}(n)$: the set of parents of n.
  while $\mathcal{N} \neq \phi$ **do**
    $n = \mathrm{argmax}_{n' \in \mathcal{N}}\, \mathit{Ord}(n')$
    // $\mathit{Ord}(n)$ is the index of $\mathit{Label}(n)$
    // in the variable order.
    $X = \mathit{Label}(n)$;
    $\mathcal{B}^1_{\boldsymbol{\theta}}[n] = \theta_{[X]}\mathcal{B}^1_{\boldsymbol{\theta}}[\mathrm{Ch}^1(n)]$
        $+\theta_{[\bar{X}]}\mathcal{B}^1_{\boldsymbol{\theta}}[\mathrm{Ch}^0(n)]$;
    $\mathcal{B}^0_{\boldsymbol{\theta}}[n] = \theta_{[X]}\mathcal{B}^0_{\boldsymbol{\theta}}[\mathrm{Ch}^1(n)]$
        $+\theta_{[\bar{X}]}\mathcal{B}^0_{\boldsymbol{\theta}}[\mathrm{Ch}^0(n)]$;
    $\mathcal{N} = \mathcal{N} \backslash \{n\} \cup \mathrm{Par}(n)$;
  end while
end

Backward and forward probabilities: We compute backward and forward probabilities like those in hidden Markov models. The procedure GetBackward() calculates *backward probabilities* for each node in the BDD representing F. A *backward probability* $\mathcal{B}^1_{\boldsymbol{\theta}}[n]$ (resp. $\mathcal{B}^0_{\boldsymbol{\theta}}[n]$) is the sum of the probabilities of all paths from node n to $\boxed{1}$ (resp. $\boxed{0}$). We set $\mathcal{B}^1_{\boldsymbol{\theta}}[\boxed{1}] = 1$ and $\mathcal{B}^0_{\boldsymbol{\theta}}[\boxed{0}] = 1$ respectively. They are calculated from terminals to the root. Contrastingly the procedure GetForward() calculates *forward probabilities* for each node from the root to terminals. A *forward probability* $\mathcal{F}_{\boldsymbol{\theta}}[n]$ is the sum of the probabilities

Procedure: GetForward()
  InitializeF();
  $\mathcal{F}_{\boldsymbol{\theta}}[root] = 1$;
  $\mathcal{N} = \{root\}$;
  while $\mathcal{N} \neq \phi$ **do**
    $n = \mathrm{argmin}_{n' \in \mathcal{N}}\ Ord(n')$;
    $X = Label(n)$;
    $\mathcal{F}_{\boldsymbol{\theta}}[\mathrm{Ch}^1(n)]\ += \mathcal{F}_{\boldsymbol{\theta}}[n]\theta_{[X]}$;
    $\mathcal{F}_{\boldsymbol{\theta}}[\mathrm{Ch}^0(n)]\ += \mathcal{F}_{\boldsymbol{\theta}}[n]\theta_{[\bar{X}]}$;
    $\mathcal{N} = \mathcal{N} \backslash \{n\} \cup \{\mathrm{Ch}^1(n), \mathrm{Ch}^0(n)\}$;
  end while
end

Procedure: GetExpectation()
  InitializeEta();
  for each $n \in \mathbf{N}$ **do**
    $X = Label(n)$;
    $e^1_n = \mathcal{F}_{\boldsymbol{\theta}}[n]\mathcal{B}^f_{\boldsymbol{\theta}}[\mathrm{Ch}^1(n)]\theta_{[X]}$;
    $e^0_n = \mathcal{F}_{\boldsymbol{\theta}}[n]\mathcal{B}^f_{\boldsymbol{\theta}}[\mathrm{Ch}^0(n)]\theta_{[\bar{X}]}$;
    $\eta^1_{\boldsymbol{\theta}}[[X]]\ += e^1_n$, k$\eta^0_{\boldsymbol{\theta}}[[X]]\ += e^0_n$;
    for each $Z \in Del^1_Y(n)$ **do**
      $\eta^1_{\boldsymbol{\theta}}[[Z]]\ += e^1_n\theta_{[Z]}$;
      $\eta^0_{\boldsymbol{\theta}}[[Z]]\ += e^1_n\theta_{[\bar{Z}]}$;
    end for
    for each $Z \in Del^0_Y(n)$ **do**
      $\eta^1_{\boldsymbol{\theta}}[[Z]]\ += e^0_n\theta_{[Z]}$;
      $\eta^0_{\boldsymbol{\theta}}[[Z]]\ += e^0_n\theta_{[\bar{Z}]}$;
    end for
  end for
end

Procedure: GetExpectation*()
  InitializeEta();
  for each $n \in \mathbf{N}$ **do**
    $X = Label(n)$;
    $e^1_n = \mathcal{F}_{\boldsymbol{\theta}}[n]\mathcal{B}^f_{\boldsymbol{\theta}}[\mathrm{Ch}^1(n)]\theta_{[X]}$
    $e^0_n = \mathcal{F}_{\boldsymbol{\theta}}[n]\mathcal{B}^f_{\boldsymbol{\theta}}[\mathrm{Ch}^0(n)]\theta_{[\bar{X}]}$
    $\eta^1_{\boldsymbol{\theta}}[[X]]\ += e^1_n$;
    $\eta^0_{\boldsymbol{\theta}}[[X]]\ += e^0_n$;
    $X' : Ord(X') = Ord(X) + 1$;
    $\zeta[X']\ += e^1_n + e^0_n$;
    $\zeta[Label(\mathrm{Ch}^1(n))]\ -= e^1_n$;
    $\zeta[Label(\mathrm{Ch}^0(n))]\ -= e^0_n$;
  end for
  $\mathcal{X} = \mathbf{X}$;
  $X = \mathrm{argmin}_{X' \in \mathcal{X}}\ Ord(X')$;
  $z = \zeta[X]$;
  $\mathcal{X} = \mathcal{X} \backslash \{X\}$;
  while $\mathcal{X} \neq \phi$ **do**
    $X = \mathrm{argmin}_{X' \in \mathcal{X}}\ Ord(X')$;
    $\eta^1_{\boldsymbol{\theta}}[[X]]\ += z\theta_{[X]}$;
    $\eta^0_{\boldsymbol{\theta}}[[X]]\ += z\theta_{[\bar{X}]}$;
    $z\ += \zeta[X]$;
    $\mathcal{X} = \mathcal{X} \backslash \{X\}$;
  end while
end

Fig. 2. Improved GetExpectation()

of all paths from the *root* to node n. The procedure InitializeF() initializes $\mathcal{F}_{\boldsymbol{\theta}}[n] = 0$ for all n.

Conditional expectations: The procedure GetExpectation() updates $\eta^x_{\boldsymbol{\theta}}[[X]]$ which is defined in Section 2 for each $X \in \mathbf{X}$. The procedure InitializeEta() sets each $\eta^x_{\boldsymbol{\theta}}[[X]] = 0$. In GetExpectation(), $f \in \{1, 0\}$ is the observed value of F, and $\mathbf{N}$ is the set of all nodes in the BDD.

Note that in order to compute probabilities properly, we need to *recover* deleted nodes. So, to denote the nodes deleted by the deletion rule, $Del^1_Y(n)$ and $Del^0_Y(n)$ are introduced in GetExpectation(). $Del^x_Y(n)$ ($x \in \{1, 0\}$) stands for the set of labels (i.e. variables) of deleted nodes between n and $\mathrm{Ch}^x(n)$. So we have $Del^x_Y(n) = \{X \in \mathbf{V}(\delta_Y) \mid Label(n) \prec X \prec Label(\mathrm{Ch}^x(n))\}$. What we actually use for the computation of conditional expectations is not GetExpectation() however, as it incurs some inefficiency, but GetExpectation*() shown in Fig. 2 which processes computation of the deleted nodes much more efficiently (details omitted).

5 Time complexities for noisy-OR models

The time complexity of building BDDs is NP-hard in general [5]. However, there are efficient techniques to build BDDs using the *Apply operation* [2] and those to find good variable orderings, be they *dynamic* or *static* [5, 6]. So building BDDs can be done efficiently in practice. In this section, we evaluate the time complexity of both building BDDs and running the BDD-EM algorithm for noisy-OR models.[4]

A noisy-OR model represents a relation between multiple causes and an effect. Let F be an observable variable representing an effect, and C_1, C_2 and C_3 basic variables representing possible causes which make F true. While the logical OR relation is represented as $F \Leftrightarrow C_1 \vee C_2 \vee C_3$, the noisy-OR relation allows for a situation where C_1 is true but F is false. For this noisy-OR model, we introduce *inhibition variables*, I_1,I_2 and I_3, which inhibit F to be true with probabilities $\theta_{[I_1]} = P(F=0 \mid C_1=1, C_2=0, C_3=0)$, $\theta_{[I_2]} = P(F=0 \mid C_1=0, C_2=1, C_3=0)$ and $\theta_{[I_3]} = P(F=0 \mid C_1=0, C_2=0, C_3=1)$, respectively. An N-input noisy-OR model between F and $C_1, C_2, \ldots, C_N$ is described by:

$$F = (C_1 \wedge \bar{I}_1) \vee (C_2 \wedge \bar{I}_2) \vee \cdots \vee (C_N \wedge \bar{I}_N).$$

Fig. 3 shows a BDD representing F under the variable ordering *Ord* such that $C_i \prec C_j$, $I_i \prec I_j$ $(i < j)$ and $C_i \prec I_k$ $(i \leq k)$. We construct a BDD from F using the Apply operation, denoted by $\text{Apply}(\delta_X, \delta_Y, \langle\text{op}\rangle)$, that builds a BDD representing $X\langle op\rangle Y$ where δ_X and δ_Y represent the boolean functions X and Y, respectively. Although the time complexity of $\text{Apply}(\delta_X, \delta_Y, \langle\text{op}\rangle)$ is $O(N_X N_Y)$ in general, where N_X (resp. N_Y) is the number of nodes in the BDD representing X (resp. Y), we can see an application of $\text{Apply}(\cdot)$ for an N-input noisy-OR model takes just $O(1)$. So the BDD is obtained by applying the Apply operation N times, and the time complexity becomes $O(N)$ under *Ord*. Also the time complexity of the E-step is $O(N)$ because $|\mathbf{N}| = 2N$ and $|\mathbf{X}| = 2N$.

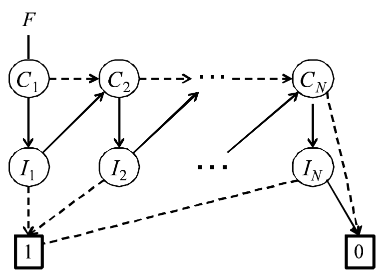

Fig. 3. A BDD representing the noisy-OR model.

[4] We confirmed the BDD-EM algorithm properly converges by numerical experiments.

6 Related work and concluding remarks

We have presented an EM algorithm that works on BDDs. Our work is considered as a succession to the previous work done by Minato et al. [7]. It shows how to compile BNs into ZBDDs to compute probabilities but probability learning is left untouched. In [3], we supplemented a necessary algorithm to apply ZBDDs to EM learning.

The introduction of BDDs solves a long-standing problem of PRISM [8], a logic-based language for generative modeling. It employs a propositionalized data structure called *explanation graph*s similar to decomposed BDDs to represent boolean formulas in disjunctive normal form. The current PRISM however assumes the *exclusiveness condition* that the disjuncts are exclusive to make sum-product probability computation possible. Since the proposed algorithms are applicable to explanation graphs as well, it allows PRISM to abolish the exclusiveness condition.

ProbLog is a recent logic-based formalism that computes probabilities via BDDs [9]. A ProbLog program computes the probability of a query atom from a disjunction of conjunctions made up of independent probabilistic atoms by converting the disjunction to a BDD and applying the sum-product computation to it. [5] Since our BDD-EM algorithm works on BDDs, integrating it with ProbLog for probability learning seems an interesting future research topic.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Computers **27**(6) (1978) 509–516
2. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
3. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the EM algorithm by BDDs. Technical Report TR08-0004, Dept. of Computer Science, Tokyo Institute of Technology (2008)
4. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. of the Royal Statistical Society B **39** (1977) 1–38
5. Drechsler, R., Sieling, D.: Binary decision diagrams in theory and practice. Int'l J. on Software Tools for Technology Transfer **3** (2001) 112–136
6. Minato, S., Ishiura, N., Yajima, S.: Shared binary decision diagram with attributed edges. Proc. ACM/IEEE Design Automation Conf. (1990) 52–57
7. Minato, S., Satoh, K., Sato, T.: Compiling Bayesian networks by symbolic probability calculation based on Zero-suppressed BDDs. In: Proc. of IJCAI'07. (2007) 2550–2555
8. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. of Artificial Intelligence Research **15** (2001) 391–454
9. De Raedt, L., Angelika, K., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discoverry. In: Proc. of IJCAI'07. (2007) 2468–2473

[5] It should be noted that a special treatment is required for the computation of conditional expectations (see [3] for details).