Discriminative Non-Parametric Learning of Arithmetic Circuits

Nandini Ramanan NANDINI.RAMANAN@UTDALLAS.EDU
Mayukh Das MXD174630@UTDALLAS.EDU
Kristian Kersting KERSTING@CS.TU-DARMSTADT.DE
Sriraam Natarajan SRIRAAM.NATARAJAN@UTDALLAS.EDU

Abstract

Arithmetic Circuits (AC) and Sum-Product Networks (SPN) have recently gained significant interest by virtue of being tractable deep probabilistic models. We propose the first gradient-boosted method for structure learning of discriminative ACs (DACs), called DACBOOST. In discrete domains ACs are essentially equivalent to mixtures of trees, thus DACBOOST decomposes a large AC into smaller tree-structured ACs and learns them in sequential, additive manner. The resulting non-parametric manner of learning DACs results in a model with very few tuning parameters making our learned model significantly more efficient. We demonstrate on standard data sets and real data sets, efficiency of DACBOOST compared to state-of-the-art DAC learners without sacrificing effectiveness.

Keywords: Probabilistic Machine Learning; Ensemble Learning; Tractable Probabilistic Models; Structure Learning.

1. Introduction

There is a recent surge in interest towards tractable probabilistic graphical models where inference is significantly more efficient [Zhao et al. (2016); Darwiche (2003); Poon and Domingos (2011)]. Of these models, both Arithmetic Circuits (AC) [Darwiche (2003)] and Sum-Product Networks (SPN) [Poon and Domingos (2011)] have garnered particular interest due to their mutual equivalence and their ability to model several other tractable models [Rooshenas and Lowd (2016)]. As pointed out by Rooshenas and Lowd[2016], most of the learning methods developed for these models are generative. So they developed and learned discriminative ACs that are a better fit for capturing log-linear models due to their ability to directly represent parameters in their nodes (as against weights in SPNs). While successful, their work has two limitations: (1) a large no. of parameters that must be tuned and (2) ACs are typically limited to being tree-structured and, hence, may break loops.

To overcome these limitations, we propose *the first non-parametric learning method for discriminative ACs (DACs)* based on gradient-boosting, called DACBOOST. Inspired by the intuition that multiple weak learners, in our case tree-structured ACs, could be more successful in learning a conditional distribution, DACBOOST introduces parameters as necessary, effectively making it more representative than a single tree-structured DAC. We derive the gradient updates that are used to reweigh the examples after each iteration and present the algorithm for learning weak, tree-structured ACs in a sequential manner. The benefits of DACBOOST are two-fold. First, it can repair broken loops by mixing different tree-structured ACs in a stage-wise manner. Second, it reduces the space of structure search and parameter updates at the same time, avoiding the seemingly difficult task of repeated full parameter estimation during structure scoring. Our extensive experiments on both

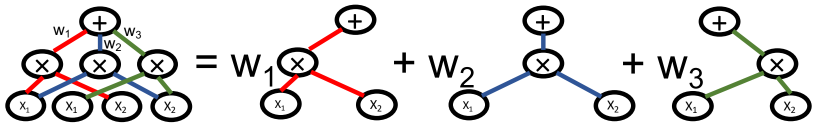

Figure 1: Figure shows how a complete and decomposable AC, here actually an SPN, is a mixture of trees. The leaves indicate univariate distributions over X_1 and X_2. Different colors highlight unique trees, which are products of univariate distributions. (Best viewed in color).

standard data sets due to Rooshenas and Lowd[2016] and on five real-world ones, DACBOOST is shown to be both effective and efficient.

We make a number of important contributions. We present the **first ensemble learning** approach for ACs. Establishing this is especially significant because (as we state later) structure learning of SPNs or ACs — for discrete domains, SPNs and ACs are equivalent [Rooshenas and Lowd (2016)] — is difficult [Zhao et al. (2016)]. Boosting reduces the space of possible structure search for learning the structure (the weak ACs) and parameter (leaves of these ACs) simultaneously, hence rendering the learning task more practical. Next, most of the prior approaches have focused on tree-structured ACs to retain tractability in learning. Triggered by the view of ACs as mixtures of trees [Zhao et al. (2016)], DACBOOST extends these tree-structured learners towards learning valid and complete ACs via boosting. Finally, our experimental evaluations on both standard and some novel and interesting real-world data clearly establish the superiority of our learner. In nearly **all the domains, we achieve equal or better performance for a fraction of the learning time** when compared to the state-of-the-art DAC learner.

2. Background

Arithmetic Circuits: There is an increased interest in tractable probabilistic models [Chandrasekaran et al. (2008)], particularly in the presence of large amounts of evidence. While other methods exist [Bouman and Shapiro (1994); Bach and Jordan (2002); Gogate et al. (2010); Karger and Srebro (2001); Osokin et al. (2011); Vernaza et al. (2008); Taskar et al. (2004); Munoz et al. (2008)], we focus on a class of methods that compile models into representations suitable for efficient inference such as (deep) probabilistic architectures like Arithmetic Circuits (ACs) [Darwiche (2003)] and Sum-Product Networks (SPNs) [Poon and Domingos (2011)]. Specifically, we consider ACs and explore an efficient learning algorithm while retaining the tractability.

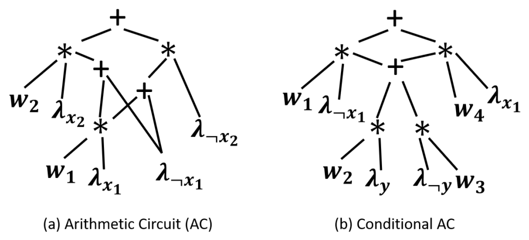

Figure 2: **Left:** Example arithmetic circuit that represents a Markov random field over two variables x_1 and x_2 and having potentials w_1 and w_2 for 2 features $x_1 \wedge x_2$ and x_2. **Right:** Conditional AC that represents distribution $P(y|x_1)$ where y is a query variable and x_1 is an evidence variable.

An *Arithmetic Circuit* $AC(\mathcal{X})$ is a rooted, directed acyclic graph over the variables $\mathcal{X}$. It contains $+$ or $*$ as internal nodes and its leaf nodes are labeled with either a non negative parameter w or an indicator λ. For an instantiation x, the value of the circuit $AC(\mathcal{X})$ is computed by assigning indicator λ_x the value 1 if $\mathcal{X}$ is compatible with x and 0 otherwise. For example, consider the simple AC in Fig 2.(a) that models the joint distribution over two binary random variables x_1 and x_2. This joint distribution is; $\lambda_{x_1}\lambda_{x_2}w_1w_2 + \lambda_{x_1}\lambda_{\neg x_2} + \lambda_{\neg x_1}\lambda_{x_2}w_2 + \lambda_{\neg x_1}\lambda_{\neg x_2}$ The complexity of evaluating an AC is linear in circuit size. Two key properties of tractable ACs [Darwiche (2003); Lowd and Rooshenas (2013)] are, (1) *Decomposibility* : An AC is decomposable *iff* $\forall$ pairs of nodes $\langle l_1, l_2\rangle$: l_1 and l_2 are children of $*$ node (product) $\Rightarrow var(l_1) \wedge var(l_2) = \emptyset$ and (2) *smoothness* : An AC is smooth *iff* $\forall$ nodes : l is a child of $+$ node n and $var(l) = var(n)$. Zhao *et al.*[2016] have shown that ACs can be viewed as mixtures of trees (Fig. 1).

Learning (Conditional) Arithmetic Circuits: Many structure learning algorithms have been proposed for discriminative SPNs such as employing an efficient SVD-approach [Adel et al.], training with the Extended Baum-Welch [Rashwan et al. (2018); Duan et al. (2020)]. Lowd and Rooshenas[2013] are the first to learn arithmetic circuits directly from data as against the expensive compilation from a different model such as Bayes Net or a Markov net. When learning discriminatively, *i.e.* $P(\mathcal{Y}|\mathcal{X})$, marginzalizing over the evidence $\mathcal{X}$ can be avoided. Rooshenas and Lowd[2016] proposed DACLEARN a discriminative learning method that learns the structure of a DAC by optimizing the penalized conditional log-likelihood (CLL). Specifically, they optimize $\log P(\mathcal{Y}|\mathcal{X})$ (where $\mathcal{X}$ is the set of evidence variables) for learning DAC. This is considerably more tractable even for large tree-width models, as opposed to typical generative training. DACLEARN, thus, iteratively grows the DAC, choosing a set of most informative features at each step of the gradient of penalized CLL. Their structure update technique employs their work on transforming MRFs to ACs [Lowd and Rooshenas (2015)]. Fig 2.(b) illustrates a Conditional AC with one query and one evidence variable. A similar approach has been proposed for discriminative learning of SPNs by Gens and Domingos[2012; 2013], an equivalent representation based on network polynomials. We consider ACs but observe that the work can be translated to SPNs as well.

Functional Gradient Boosting (FGB) transforms the problem of learning a conditional distribution to learning a sequence of function approximation problems (following the work by Friedman[2001]). Thus a conditional distribution is represented as a weighted sum of regression models learned sequentially via a stage-wise optimization. For a particular example, say y_i, its conditional distribution given its parents $\mathbf{x}_i$ can be learned by fitting a model $P(y|\mathbf{x}) \propto e^{\psi(y,\mathbf{x})}$. The key insight is that instead of learning in the parameter space (P), gradient is obtained in the functional-space ψ. Now, FGB successively approximate ψ as a sum of weak learners, which are typically regression trees. Staring from an initial ψ_0 functional gradient ascent iteratively adds gradients Δ_i. For every iteration i a new weak model h_i is fitted to the gradient. After m iterations, the potential is given by $\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m$. Here, Δ_m is the functional gradient at step m,

$$\Delta_m = \eta_m \cdot E_{\mathbf{x},y}\left[\frac{\partial}{\partial \psi_{m-1}} \log\ P(y \mid \mathbf{x};\ \psi_{m-1})\right]$$

where η_m is the learning rate. Dietterich *et al.*[2004] was the first to train a probabilistic model, a CRF by evaluating the gradient at each step for every training example and fitting a regression tree h_m to these derived examples. $[(x_i, y_i), \Delta_m(y_i; x_i)]$ is a close and reasonable approximation of the desired Δ_m and essentially, points in the same direction serving as a good approximation of the true functional gradient. This has been effectively adapted for several probabilistic models [Khot et al. (2011); Ramanan et al. (2018)].

3. Learning Discriminative ACs

Given Data set $\mathcal{D}(\mathcal{X}, \mathcal{Y})$, where $\mathcal{Y}$ is a set of query variables, & $\mathcal{X}$ is a set of evidence variables, **find** the structure and parameters of a discriminative arithmetic circuit (DAC), i.e., the distribution $P(\mathcal{Y}|\mathcal{X})$.

Unlike generative ACs, DACs allow conditioning the query variables over evidence, allowing them to be trained similar to CRFs [Rooshenas and Lowd (2016)]. DACs offer similar benefits as CRFs in modeling complex dependencies between evidence and query while retaining tractability for learning and inference. Given that gradient-boosting is state-of-the-art in learning CRFs [Dietterich et al. (2008); Chen et al. (2015)], we derive a learning algorithm based on gradient-boosting for full model learning (structure + parameter learning) of DACs.

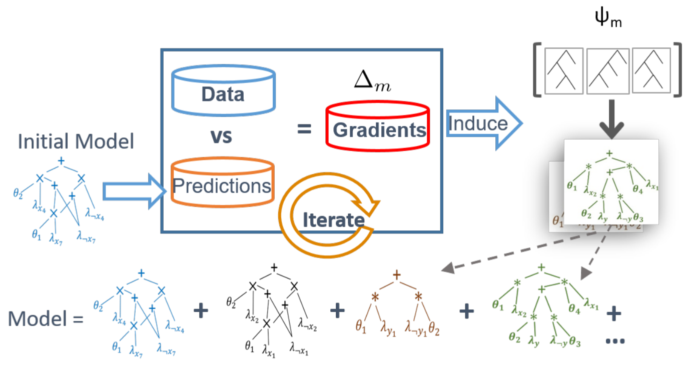

Figure 3: Learning of Discriminative Arithmetic Circuits via boosting. At each iteration a small weak DAC is learned for each query variable. Once a DAC is learned at each iteration, the weights of examples are computed and a new AC is learned. These are then added to the model and the process continues until convergence.

Gradient Boosting for Conditional ACs: We adopt the following convention for notation - subscript i denotes the i^{th} example, superscript j denotes the j^{th} feature of the evidence set and superscript (p) denotes the p^{th} query variable. Given this notation, following DACLEARN (Lowd et al. 2016) that optimizes conditional log-likelihood of train data set $\mathcal{D}$ by finding the best set of features f, where, $\mathbf{y} = \mathcal{Y}^{(p)} \subseteq \mathcal{Y}$ and $\mathbf{x} \subseteq \mathcal{X}$, we define $CLL(\mathcal{D}) :=$

$$\sum_{(\mathbf{y},\mathbf{x})\in\mathcal{D}} \log P(\mathbf{y}|\mathbf{x}) = \sum_{(\mathbf{y},\mathbf{x})\in\mathcal{D}} \sum_{j} w^j f^j - \log Z(\mathbf{x})$$

where Z is the normalization constant (partition function). Recall that FGB obtains point-wise gradient for each example separately. So now, the CLL for example i with query variable y (we drop the superscript for brevity) is,

$$\begin{aligned}\log P(y_i = 1|\mathbf{x}_i) &= \sum_j w^j f^j(y_i = 1|\mathbf{x}_i) - \log Z(\mathbf{x}_i)\\ &= \sum_j w^j f^j(y_i = 1|\mathbf{x}_i) - \log \sum_{y'} \exp\big(\sum_j w^j f^j(y_i = y'|\mathbf{x}_i)\big)\end{aligned}$$

Like CRFs, the definition of DACs naturally allow for the functional representation. $\psi(y_i|\mathbf{x}_i)$, the potential of y_i given $\mathbf{x}_i$ for i^{th} example can be defined as, $\psi(y_i|\mathbf{x}_i) = \sum_j w^j f^j$. Note that while we explain the process assuming binary query variables, they can be easily extended to the multivalued case. While learning a binary discriminative model for $y_i = 1$,

$$\log P(y_i = 1|\mathbf{x}_i) = \psi(y_i = 1|\mathbf{x}_i) - \log \sum_{y'} \exp(\psi(y_i = y'|\mathbf{x}_i)))$$

Now computing the point-wise derivative of CLL w.r.t ψ $\forall i$;

$$\frac{\partial P(y_i|\mathbf{x}_i)}{\partial \psi(y_i|\mathbf{x}_i)} = I(y_i|\mathbf{x}_i) - P(y_i = 1|\mathbf{x}_i)$$

Note that the above expression is similar to the gradients for well-known probabilistic models [Dietterich et al. (2008); Natarajan et al. (2012)]. The weight of each example is simply the difference between whether the example is true according to the data (denoted by I) and the predicted probability of the query being true according to the current model. These gradients are then used in the next iteration of boosting where a weak learner (DAC in our case) is learned to fit these point-wise gradients. The score of the structure using the set of candidate features $\mathcal{F}$ can be rewritten as: $Score(\mathcal{F}) = \Delta_{cll}(\mathcal{F})$, where Δ_{cll} denotes the change in CLL. The goal is to identify the features that maximize this change in CLL.

A key aspect of our learning approach is that, unlike in the approach due to Rooshenas and Lowd[2016], there is **no necessity to introduce an explicit model complexity penalty**. Learning weak models automatically takes care of regularization by controlling depth of the learned ACs. Specifically, given functional-gradients for each example, we learn a small, tree-structured AC by searching through the space of potential features to add next by minimizing the weighted variance of conditional distribution according to current model. Once AC is constructed, next step is to estimate weights w^j at leaves of the DAC. To estimate them, we maximize increment in CLL function:

$$\Delta_{cll}(\mathcal{F}) = \sum_{(\mathbf{y},\mathbf{x})\in D} \sum_{j} w^j \hat{P}(f^j|\mathbf{x}) - \Delta \log Z(\mathbf{x})$$

where $\hat{P}$ is the new empirical probability distribution after introducing the new candidate features. The gradient is:

$$\frac{\partial \Delta_{cll}(F)}{\partial w^j} = \sum_{(\mathbf{y},\mathbf{x})\in D} \frac{\exp{(w^j)} P(f^j|\mathbf{x})}{\exp{(\Delta \log Z(\mathbf{x}))}}$$

Learning Joint Models: When learning with multiple queries, following prior work on pseudo-likelihood training of MRFs [Besag (1975); Richardson and Domingos (2006); Khot et al. (2011)], we factorize the joint over the queries as product of conditionals, given an ordering. More precisely, given a set of queries $\mathbf{y}_{1:k}$, an ordering $k < k-1 < ...1$ and the evidence $\mathbf{x}$, the joint becomes,

$$P(\mathbf{y}_{1:k} \mid \mathbf{x}) = \Pi_{i=2}^{k} P(y_i|y_1, ..., y_{i-1}, \mathbf{x}) \cdot P(y_1|\mathbf{x})$$

We boost each conditional separately (& in parallel) while training. During testing, we perform Gibbs sampling over unobserved variables [Khot et al. (2011)] and report joint likelihood of test set.

The DACBOOST Algorithm: Alg. 1 summarizes our boosting approach. Here DACBOOST() is the primary procedure that learns an ensemble of gradient boosted Arithmetic Circuits for a given query variable $\mathcal{Y}^{(p)}$ and training data $\mathcal{D}$. As explained earlier, $\mathcal{X}$ and $\mathcal{Y}$ are sets of evidence and query variables respectively, and so, a training example $\mathcal{D}_i = \langle \mathcal{X}_i, \mathcal{Y}_i \rangle$. The argument p is the index to the particular query variable in $\mathcal{Y}$ for which the discriminative model will be learned. For a collective classification task the function is called successively for each query variable. Since there could potentially be multiple query variables, we denote the current query variable as $\mathcal{Y}^{(p)}$.

In the m^{th} iteration of functional-gradient boosting, we compute the functional gradients for these examples using the current model F_m and the evidences of $\mathbf{y}$ as per this model (line 5). The gradients $S_m^{(p)} = \{\langle y_i, \Delta_i \rangle\}$ (where Δ_i is the gradient value for the example y_i) are then used to learn a new weak AC ψ_m and added to the model **[lines 6,8]**. Note, however, that SUBSAMPLENEG() **[line 6]** sub-samples from the negative examples and the corresponding gradients and evidences ($\mathcal{Y}' \subset \mathcal{Y}^{(p)} = \{y : \forall y, y \in \mathcal{Y}^{(p)}, y \in \mathcal{Y}', y = 0, y$ consistent with sampler$\}$, $S' = \{s_i : \forall i, s_i \in S_m^{(p)}, y_i \in \mathcal{Y}'\}$ and $\mathcal{X}' = \{\mathbf{x}_i : \forall i, \mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}'\}$). LEARNWDAC() learns a new weak AC based on the current sub-sampled gradients as summarized in Alg. 2.

Algorithm 1 DACBOOST: Boosted Arithmetic Circuits

```
Require: $\mathcal{X}, \mathcal{Y}, \mathcal{D}, \mathbf{P}$
 1: Initialize to uniform prior: Model $F_0 \leftarrow \gamma$
 2: Learn upto M gradient steps: $m = 1$
 3: repeat
 4:     $F_m \leftarrow F_{m-1}$
 5:     repeat
 6:         $S_m^{(p)} \leftarrow$ ComputeGradients$\mathcal{Y}^{(p)}, \mathcal{X}, F_m$
 7:         Gradients $\{\Delta_i\}_{i=1}^{|\mathcal{D}|}$, $\forall$ examples $y_i \in \mathcal{Y}^{(p)}$
 8:         $S', \mathcal{X}', \mathcal{Y}' \leftarrow$ SubSampleNeg$S_m^{(p)}, \mathcal{X}, \mathcal{Y}^{(p)}$
 9:         $\psi_m \leftarrow$ LearnWDAC$S', \mathcal{X}', \mathcal{Y}'$
10:         $F_m^p \leftarrow F_(^p m - 1) + \psi_m(p)$
11:     until $p \in \mathbf{P}$ Iterate through Query predicates
12: until $m = M$
13: Return $F_m$
```

Alg. 2 outlines the procedure LEARNWDAC, that takes the gradients and the corresponding evidence and labels (for the query variable $\mathcal{Y}$) as input. The model and the supporting data structures are initialized **[lines 2-5]**. Note that $\mathbf{w}$ refers to all the parameter nodes in a DAC. fs is a min-heap data structure used later to search for the best set of features to be considered in constructing the DAC. To learn a valid conditional weak AC to fit the gradients at the current step we have to determine the best (most informative) feature set and it is done in a step-wise fashion. First, a set of regression trees (rooted at every feature/variable in $\mathcal{X}$) are learned using weighted variance as the scoring function **[lines 6-9]**. The feature set $\mathcal{F}$ is then computed by first constructing a min-heap fs with the features from the learned regression trees, using their weighted-variance scores. We search and retrieve from min-heap, the edges with weights lower than Ω times the highest weighted variance as seen from the current feature set given by $threshold = \Omega \cdot \max_{T \in \mathbb{T}} Score(T)$ **[lines 10,11]**. $\mathcal{F}$ is used to construct a DAC $C_{\mathbf{w}}$, by iteratively scanning each feature f_k in the feature set and including it in the AC if the change in conditional log-likelihood $\Delta_{cll}(f_k)$ is above a positive threshold τ ($\tau > 0$) **[lines 12-17]**. Whenever a feature f_k is included in the DAC, we jointly optimize the parameters $\mathbf{w}$ after updating the initial structure. We use L-BFGS to optimize the weights in our model. Note that, for constructing/updating the AC $C_{\mathbf{w}}$ **[line 14]** we utilize the *'Split AC'* approach proposed in the ACMN algorithm [Lowd and Rooshenas (2013)].

Discussion — DACBOOST closes loops by inducing wide-and-deep DACs: Before moving on to our empirical evaluation, let us discuss some key insights about DACBOOST. DACBOOST differs from the currently best discriminative learning algorithm DACLEARN [Rooshenas and Lowd (2016)]. While DACLEARN induces tree structured ACs, ours is capable of learning DAGs. Intuitively, their approach can be viewed as breaking some loops in the true generative model. Since we boost the learning, we "overlay" several trees and hence could potentially repair some loops, that may otherwise have been broken if a single tree-structured AC was learned. Exploring the connection to tree-reweighted bounds and/or stacking learning to deeply understand the properties of our learning algorithm is an interesting future direction. Also, we are strongly motivated by the observation that complete and valid SPNs (correspondingly ACs) can be induced by a mixture of trees [Zhao et al. (2016)]. Thus, our work can be seen as extending tree SPN and AC learners with boosting to learn valid DACs with the observation that valid SPNs (and ACs) are additive tree models. That is, we widen the deep DACs by boosting in features and feature combinations as needed. We verify this

Algorithm 2 LEARNWDAC: Fit Weak Conditional AC

```
Require: $S, \mathcal{X}, \mathcal{Y}^{(p)}$
 1: The gradient set $S = \{\langle y_i, \Delta_i \rangle\}$
 2: Conditional Arithmetic Circuit $C_{\mathbf{w}} \leftarrow \emptyset$
 3: Initialize empty AC; $\mathbf{w}$: set of parameters
 4: Initialize min-heap $fs \leftarrow \emptyset$
 5: Feature set $\mathcal{F} \leftarrow \emptyset$
 6: Regression Tree set $\mathbb{T} \leftarrow \emptyset$
 7: repeat
 8:     $T_j \leftarrow$ Regression Tree rooted at $f_j$
 9:     Scored using Weighted-Variance
10:     $\mathbb{T} \leftarrow \mathbb{T} \cup T_j$
11: until $\forall$ features $f_j \in \mathcal{X}$
12: $fs \leftarrow$ Min-Heap$f_j : \forall\, T_j \in \mathbb{T}$, Scores
13: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$: select best set $\{f\} \equiv \{T\} \subset \mathbb{T}$
14: min-heap search on $fs$ w/ threshold $\Omega * max$
15: repeat
16:     if $\Delta_{cll}(f_k) > \tau$ then
17:         Update $C_{\mathbf{w}}$ with $f_k$
18:         Update parameters $\mathbf{w}$
19:     end if
20: until each $f_k$ in $\mathcal{F}$
21: Return $C_\theta$
```

empirically by demonstrating that our DACBOOST algorithm learns effective DACs by achieving equal or better performance in nearly all the domains as the state-of-the-art DACLEARN algorithm. By virtue of learning shallower models, our approach is significantly faster.

4. Empirical Evaluation

We aim to answer the following questions explicitly in our experimental evaluations:

(Q1) Is DACBOOST effective against the state-of-the-art?

(Q2) Is DACBOOST more efficient than a single DAC?

(Q3) Can DACBOOST handle joint learning effectively?

We implemented DACBOOST as an extension to the DACLEARN code-base available as a part of the open-source Libra toolkit[1] [Lowd and Rooshenas (2015)]; consequently, DACBOOST inherits all the system requirements and the library dependencies as the original DACLEARN. We evaluated DACBOOST against the state-of-the-art discriminative structure learning algorithm for ACs, DACLEARN, on both real and standard data sets. All experiments were conducted on *Intel(R) Xeon(R) CPU E5-2630 v3* server machines, clocking @ 2.40GHz and usable memory of 235GB.

Data Sets Description: We investigated **(Q1)**, **(Q2)** on both standard domains and some novel domains for probabilistic modeling. To answer **(Q3)**, we demonstrate the performace of DACBOOST on a few standard domains with $10\% - 20\%$ of the features as query variables. Specifically, we

1. http://libra.cs.uoregon.edu/

chose four clinical/medical data sets, a network traffic for DDoS attack detection data set and some standard data sets. We now describe the data sets briefly focusing on the novel ones:
1. Alzheimer's: The Alzheimer's Disease NeuroIntiative (ADNI[2]) is designed to verify whether MRI and PET images, genetics, cognitive tests and blood biomarkers can be used for early prediction Alzheimer's disease. We learn DAC for modeling Alzheimer's vs cognitively normal, conditioned on demographics features and MMScore (cognitive test score) ($\#numVars = 29, \#numEx = 350$).
2. Drug-Drug interactions (DDI): This data set consists of 78 drugs obtained from (DrugBank[3]). The goal is to learn a distribution of drug-drug interactions conditioned on the chemical pathways [Dhami et al. (2018)]$\#numVars = 25, \#numEx = 16k$).
3. DDoS attack detection (DDoS): Employed in the work of Ricks et al.[2018], benign and large-scale botnet network traffic is captured for use in DDoS attack detection. A key aspect is the automation of client-side human behavior for generation of benign network traffic in a manner scalable to network size. Our goal is to learn a model for attack modeling given benign and botnet network traffic $\#numVars = 20, \#numEx = 33650$).
4. Post-Partum Depression (PPD): Inspired by the work of Natarajan et al[2017], the goal is to model post-partum depression diagnosis based on online questionnaire data including demographics, family history (relationship), social support, economic status, infant behavior and CDC questions$\#numVars = 66, \#numEx = 130$).
5. Parkinson's: Parkinson's Progression Markers Initiative (PPMI[4]) is a study designed to identify biomarkers that impact Parkinson's progression in a subject. Features include imaging data, clinical data, demographics and Montreal Cognitive Assessment Score (MoCA) and the goal is to learn conditional distribution of occurrence of PPMI$\#numVars = 119, \#numEx = 1680$).

We also employed the benchmark data sets that were extensively used in prior work on learning SPNs and ACs [Gens and Domingos (2013); Rooshenas and Lowd (2014, 2016)]. The goal is to evaluate on benchmarks where DACLEARN is the state-of-the-art as well as on **novel, real** domains.

Experimental Protocol: Following the experimental protocol of the previous work [Rooshenas and Lowd (2016)], we created train and test sets (with $80-20$ split). For DACLEARN, we used the parameterization as suggested in the paper with L1 prior of 0.1, 0.5, 1, and 2, and feature penalties of 2, 5, and 10, and an edge penalty of 0.1, a maximum circuit size of 1M edges, and a feature batch size of 2. For DACBOOST, based on the training conditional log-likelihood, we chose the number of weak DACs to be between 3 and 12.

Results: The performance metrics and running times on the 5 novel data sets are summarized in Table 1. It can be observed that DACBOOST is at least as good as or better than DACLEARN in several of these data sets ($4/5$). More importantly, it is faster in the majority of the data sets and in ADNI $2x$ faster. This answers both **(Q1)** and **(Q2)** affirmatively making it an attractive modeling choice for real data. To understand the differences, let us focus on specific data sets. Consider the lower performance in `DDI` data set which is a 3-D image that is sparse. Several feature values are 0 across all the examples. Our hypothesis is that such features are not specifically useful for constructing weak ACs and hence boosting does not perform as well (even though it is not significantly worse). In `DDoS` data set where the performance is comparable, many of the examples are repeated and the amount of these repetitions is high. In DACBOOST, in Alg. 2, it can be observed

2. www.loni.ucla.edu/ADNI
3. https://www.drugbank.ca/
4. www.loni.ucla.edu/PPMI

Data sets	DACBOOST				DACLearn				Speedup
	CLL	Time	Avg Edges	Avg Depth	CLL	Time	Edges	Depth	
ADNI	**-0.178**	**0.950**	180	5	-0.180	1.90	329	10	2.000
DDI	-0.264	**133.87**	490	8	**-0.245**	158.86	1981	18	1.186
DDoS	**-0.018**	133.52	273	10	-0.019	**121.11**	483	14	0.907
PPD	**-0.411**	**12.19**	476	6	-0.785	13.83	819	8	1.134
PPMI	**-0.163**	**353.74**	980	7	-0.188	522.99	2429	20	1.478

Table 1: Evaluation results of DACBOOST on real domains (data sets). Conditional log-likelihood, *CLL*, illustrates the effectiveness (higher ⇒ better), while the running time, *Time(Sec.)* shows the efficiency (lower ⇒ better). *Speedup* is the ratio of the running time of DACLEARN to that of DACBOOST. To demonstrate the structure complexity of the learned models across all the data sets, we present the average number of edges and the average depth of each small DAC, for DACBOOST. For DACLearn, we present the model size.

Data sets	DACBOOST				DACLearn				Speedup
	CLL	Time	Avg Edges	Avg Depth	CLL	Time	Edges	Depth	
NLTCS	**-0.148**	**19.68**	252	8	-0.152	40.35	861	20	2.050
MSNBC	**-0.108**	**144.400**	399	10	-0.110	12153.90	10010	24	84.167
KDDCup	**-0.010**	2612.565	728	8	-0.073	**1248.77**	1904	22	0.478
Plants	**-0.069**	**560.368**	763	8	-0.255	13077.74	7497	24	23.337
Audio	**-0.459**	**3619.24**	980	8	-0.483	18383.51	5957	16	5.07
Jester	**-0.493**	**715.53**	980	8	-0.665	9851.41	6244	14	13.768
Netflix	**-0.554**	**3150.70**	980	8	-0.642	21588.92	6895	16	6.85
Accident	**-0.174**	**1646.87**	1057	8	-0.2776	22734.73	6923	26	13.81
Retail	**-0.087**	3246.29	1225	8	-0.090	**2958.99**	2191	24	0.911
Pumsbstar	**0.00**	1017.60	932	6	**0.00**	**720.98**	1218	8	0.708
DNA	**-0.346**	**350.56**	1540	8	-0.801	1030.03	3129	18	2.938
Kosarek	**-0.016**	**3300.885**	1610	8	-0.021	3778.92	2114	18	1.144
MSWeb	-0.118	**1132.09**	2065	6	**-0.109**	13996.87	4921	30	12.363
Movie	**-0.358**	**3720.31**	3780	8	-0.444	20526.80	7665	24	5.517
WebKB	**-0.178**	**6154.40**	6013	7	-0.293	17840.81	10630	18	2.900
R-52	**-0.126**	**26606.96**	6503	8	-0.138	30147.31	7392	10	1.12
20 NG	**-0.267**	21680.70	6650	8	-0.274	**20487.39**	7035	8	0.944
BBC	**-0.084**	11744.93	7686	8	-0.127	**5596.90**	7742	10	0.476
Ad	**0.0**	**3876.07**	10955	6	0.0	8087.36	10955	6	2.086

Table 2: Results of DACBOOST on benchmark data sets that have been used and reported in DACLEARN. Conditional log-likelihood, *CLL*, shows the effectiveness (higher ⇒ better), while the running time, *Time(Sec.)* shows the efficiency (lower ⇒ better). We present the average number of edges and the average depth of each small DAC, for DACBOOSTand the model size of the resulting model for DACLearn. It can be seen that the total number of edges in many of the domains are comparable but DACBOOST is significantly faster to learn.

that we sub-sample the negatives. This subsampling process, with large number of repetitions, can potentially be ineffective, thus explaining the slower convergence rate. Exploring intelligent sampling of negatives is an interesting direction. In data sets with no missing or repeated algorithms, both algorithms perform similarly but Boosting is clearly faster. Over all, DACBOOST performs equally or better than the strong baseline in majority of the data sets.

To further evaluate, we considered benchmark data sets (Table 2). The first key observation across **all** data sets is that, DACBOOST *is significantly faster than state-of-the-art* in most cases. In domains where it is worse, the efficiency is significantly higher (about 12 times faster) for a small loss in CLL. This allows us to answer **(Q2) strongly** affirmatively. To further understand benefits of boosting, we analyze size of models learned in Tab. 2. We present number of edges and depth of the baseline DACLEARN and *average* no. of edges and depth per DAC for the DACBOOST. Since our model explicitly learns weak DACs in an additive fashion, it is not surprising that both the depth and the number of edges of the induced DACs are significantly smaller on average. However, even the total number of edges for the entire boosted model is in many cases still smaller than the full DAC induced by DACLEARN. It must be mentioned that DACBOOST does not have many parameters and they are introduced as needed. Hence, we do not explicitly use any regularization techniques except to control the number of ACs learned. Finally, even in the cases where the model sizes are comparable, DACBOOST appears to be significantly faster demonstrating the efficiency of the proposed approach. To further answer **(Q1)**, when observing the CLL values in the table, it can be easily observed that the boosting approach is equal or better in nearly **all** the domains. It can be stated that DACBOOST is indeed more efficient (**with speedups ranging b/w** 1.12 **to** 84) & equally effective when compared to learning single ACs. For joint learning evaluation, we ran on 3 standard data sets, with 20% percentage of variables being chosen as query. We assumed that all the query variables are unobserved and hence ran an approximate gibbs sampler to perform inference. More principled inference algorithm for joint learning is another interesting future direction. As with the single variable case, we evaluated on a test set and present the CLL. Note that, since we learn each conditional in parallel, our learning time is the worst-case of all conditionals, as opposed to the baseline which learns them sequentially. Thus in all data sets, DACBOOST is significantly (orders of magnitude) faster. It can be observed that test set CLL is also better in 2 of the 3 domains, thus answering **(Q3)** affirmatively. A similar behavior has been observed in case of dependency networks and its relational extension [Natarajan et al. (2012)] when compared to learning a single MRF.

	CLL		
	NLTCS	MSNBC	KDDCup
DACBOOST	**-0.709**	-0.886	**-0.519**
DACLEARN	-0.788	**-0.861**	-0.563

Figure 4: Conditional log-likelihood (CLL) comparison for joint learning on benchmark domains.

In summary, our evaluations on novel and established benchmark data sets **clearly demonstrate the potential for boosting DACs**. Even in domains where there is a marginal loss in performance, the **learning time is significantly smaller** and most importantly, boosting can be parallelized during joint learning. Another advantage of ensemble learning is that, since the learned ACs are typically weak, regularization is simple and effective and is handled implicitly.

5. Conclusions

ACs emphasize the important role of depth in learning tractable probabilistic models. We argue that width is equally important. Unfortunately, wide and deep probabilistic models are more difficult to train. We presented the first boosting framework to ease the training of tractable discriminative probabilistic models, specifically conditional ACs. We derived the functional gradients of the examples, outlined the method for learning weak ACs and presented the algorithm, called DACBOOST, for learning them given the data. Our empirical evidence shows that boosted conditional ACs can gain predictive performance, sometimes in an fraction of time.

There are several avenues for future work: Analysis of the theoretical properties including bounds and convergence is an immediate future direction. Using domain-specific human input as an inductive bias could make the algorithm converge even faster. Here, carrying over ideas of residuals networks [He et al. (2016)] to ACs appears to be promising. Finally, how to make a broader class of tractable probabilistic models including generative models wider and deeper remains an open question from both a theoretical as well as an algorithmic perspective.

Acknowledgement

The authors thank the anonymous reviewers. SN and NR gratefully acknowledge the support of AFOSR award FA9550-18-1-0462 and the support of relationalAI. KK acknowledges the support of the DFG project CAML (KE 1686/3-1, SPP 1999) and of the RMU project DeCoDeML. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the AFOSR,or the US government.

References

T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an svd-based algorithm.

F. R. Bach and M. I. Jordan. Thin junction trees. In *NIPS*, 2002.

J. Besag. Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 1975.

C. A. Bouman and M. Shapiro. A multiscale random field model for bayesian image segmentation. *IEEE Transactions on image processing*, 1994.

V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of inference in graphical models. In *UAI*, 2008.

T. Chen, S. Singh, B. Taskar, and C. Guestrin. Efficient second-order gradient boosting for conditional random fields. In *AISTATS*, 2015.

A. Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 2003.

D. S. Dhami, G. Kunapuli, M. Das, D. Page, and S. Natarajan. Drug-drug interaction discovery: Kernel learning from heterogeneous similarities. *Smart Health*, 2018.

T. Dietterich, A. Ashenfelter, and Y. Bulatov. Training CRFs via gradient tree boosting. In *ICML*, 2004.

T. Dietterich, G. Hao, and A. Ashenfelter. Gradient tree boosting for training conditional random fields. *JMLR*, 2008.

H. Duan, A. Rashwan, P. Poupart, and Z. Chen. Discriminative training of feed-forward and recurrent sum-product networks by extended baum-welch. *International Journal of Approximate Reasoning*, 124:66–81, 2020.

J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 2001.

R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *NIPS*, 2012.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *ICML*, 2013.

V. Gogate, W. Webb, and P. Domingos. Learning efficient markov networks. In *NIPS*, 2010.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

D. Karger and N. Srebro. Learning markov networks: Maximum bounded tree-width graphs. In *SODA*, 2001.

T. Khot, S. Natarajan, K. Kersting, and J. Shavlik. Learning Markov logic networks via functional gradient boosting. In *ICDM*, 2011.

D. Lowd and A. Rooshenas. Learning markov networks with arithmetic circuits. In *AISTATS*, 2013.

D. Lowd and A. Rooshenas. The libra toolkit for probabilistic models. *JMLR*, 2015.

D. Munoz, N. Vandapel, and M. Hebert. Directional associative markov network for 3-d point cloud classification. In *3DPVT*, 2008.

S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *MLJ*, 2012.

S. Natarajan, A. Prabhakar, N. Ramanan, A. Baglione, K. Connelly, and K. Siek. Boosting for postpartum depression prediction. In *CHASE*, 2017.

A. Osokin, D. Vetrov, and V. Kolmogorov. Submodular decomposition framework for inference in associative markov networks with global constraints. In *CVPR*, 2011.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *UAI*, 2011.

N. Ramanan, G. Kunapuli, T. Khot, B. Fatemi, S. M. Kazemi, D. Poole, K. Kersting, and S. Natarajan. Structure learning for relational logistic regression: An ensemble approach. In *KR*, 2018.

A. Rashwan, P. Poupart, and C. Zhitang. Discriminative training of sum-product networks by extended baum-welch. In *International Conference on Probabilistic Graphical Models*, pages 356–367, 2018.

M. Richardson and P. Domingos. Markov logic networks. *MLJ*, 2006.

B. Ricks, B. Thuraisingham, and P. Tague. Lifting the smokescreen: Detecting underlying anomalies during a ddos attack. In *ISI*, 2018.

A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *ICML*, 2014.

A. Rooshenas and D. Lowd. Discriminative structure learning of arithmetic circuits. In *AISTATS*, 2016.

B. Taskar, V. Chatalbashev, and D. Koller. Learning associative markov networks. In *ICML*, 2004.

P. Vernaza, B. Taskar, and D. D. Lee. Online, self-supervised terrain classification via discriminatively trained submodular markov random fields. In *ICRA*, 2008.

H. Zhao, P. Poupart, and G. J. Gordon. A unified approach for learning the parameters of sum-product networks. In *NIPS*, 2016.