Learning Logistic Circuits

Yitao Liang
Computer Science Department
University of California, Los Angeles
yliang@cs.ucla.edu

Guy Van den Broeck
Computer Science Department
University of California, Los Angeles
guyvdb@cs.ucla.edu

Abstract

This paper proposes a new classification model called *logistic circuits*. On MNIST and Fashion datasets, our learning algorithm outperforms neural networks that have an order of magnitude more parameters. Yet logistic circuits have a distinct origin in symbolic AI, forming a discriminative counterpart to probabilistic-logical circuits (ACs, SPNs, PSDDs). We show that parameter learning for logistic circuits is convex, and that a simple local search algorithm can induce strong model structures from data.

1 Introduction

Circuit representations are a promising synthesis of symbolic and statistical methods in AI. They are "deep" layered data structures with statistical parameters, yet they also capture intricate structural knowledge. Recently, many target representations have been proposed for learning discrete probability distributions (e.g., ACs [7], Weighted SDD [1], PSDD [5], Cutset Networks [10] and SPNs [9]). Collectively, these approaches achieve the state of the art in density estimation and vastly outperform classical probabilistic graphical models learners [6]. However, it is rare to observe the same success when deploying circuit representations for classification and discriminative learning. Probabilistic circuit classifiers lag behind the performance of neural networks [2].

In this paper, we propose a new classification model called *logistic circuits*, which shares many syntactic properties with the target representations mentioned earlier. One can view logistic circuits as the discriminative counterpart to probabilistic circuits. Owing to their elegant properties, learning the parameters of a logistic circuit can be reduced to a logistic regression problem and is therefore convex. Learning logistic circuit structure is reduced to a simple local search problem using primitives from the probabilistic circuit learning literature [6].

We run experiments on standard image classification benchmarks (MNIST and Fashion) and achieve accuracy higher than much larger MLPs and even CNNs. For example, logistic circuits obtain 99.4% accuracy on MNIST. Furthermore, we show our learner is very data efficient, managing to still learn well with limited data.

2 Representation

This section introduces the logistic circuit representation.

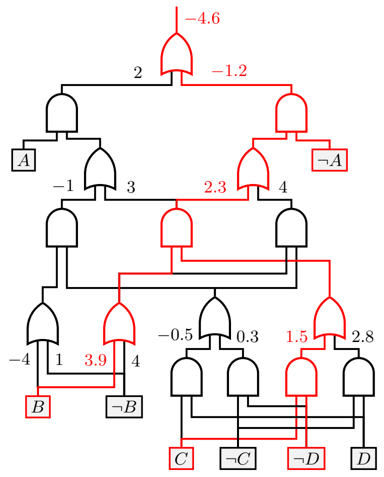
Notation An uppercase X denotes a Boolean random variable and a lowercase x denotes a specific assignment to X. We use Boolean random and logical variables interchangeably. A set of variables $\mathbf{X}$ and its joint assignment $\mathbf{x}$ are denoted in bold. A complete assignment $\mathbf{x}$ to all variables is a possible world, or interchangeably a data example. Literals are variables or their negation. Logical sentences are constructed from literals and connectives such as AND and OR in the usual way. An assignment $\mathbf{x}$ that satisfies a logical sentence α is denoted $\mathbf{x} \models \alpha$.

2.1 Logical Circuits

A logical circuit is a directed acyclic graph representing a logical sentence, as depicted in Figure 1a (ignoring parameters for now). Each inner node is either an AND gate or an OR gate.[1] A leaf (input) node represents a Boolean literal, that is, X or $\neg X$, where the node can only be satisfied if X is set to 1 (true) respectively 0 (false).

The following properties are key for logical circuits to be well-behaved [3]. An AND gate is *decomposable* if its inputs depend on disjoint sets of variables. For example, the top-most AND gates in Figure 1a depend on A in one input and on $\{B, C, D\}$ in their other input. In this

[1] We consider negation-normal-form circuits where no negation is allowed except at the leafs/inputs [3].



(a) Logistic circuit

A	B	C	D	$\Pr(Y = 1 \mid ABCD)$
1	0	1	1	4.31%
0	1	1	0	86.99%
1	1	1	0	99.70%

(b) Classification probabilities for select examples

Figure 1: A logistic circuit with example classifications.

paper, we assume every AND gate has two inputs and call its left input prime and right input sub. An OR gate is *deterministic* if for any single complete assignment, at most one of its inputs can be set to 1. For example, the left input to the root OR gate in Figure 1a is 1 precisely when $A = 1$, and its other input is 1 precisely when $A = 0$.

Logical circuits can be extended to *probabilistic circuits* that represents a probability distribution over binary random variables, for example by parameterizing wires with conditional distributions [5]. Probabilistic circuits have been successfully used for generative learning [6].

2.2 Logistic Circuits

This paper proposes *logistic circuits* for classification. Syntactically, they are logical circuits where every AND is decomposable and every OR is deterministic. However, logistic circuits further associate real-valued parameters $\theta_1, \ldots, \theta_m$ with the m input wires to every OR gate, as well as a parameter with the root output wire of the entire circuit. For example, the root OR node in Figure 1a associates parameters 2 and -1.2 with its two inputs.

To give semantics to logistic circuits, we first characterize how a particular complete assignment $\mathbf{x}$ (one data example) propagates through the circuit.

Definition 1 (Boolean Flow)**.** *Consider a deterministic OR gate n. The Boolean flow $f(n, \mathbf{x}, c)$ of a complete assignment $\mathbf{x}$ between parent n and child c is*

$$f(n, \mathbf{x}, c) = \begin{cases} 1 & \textit{if } \mathbf{x} \models c \\ 0 & \textit{otherwise} \end{cases}$$

For example, under the assignment $A = 0, B = 1, C = 1, D = 0$, the root node in Figure 1a has a Boolean circuit flow of 0 with its left child and 1 with its right child. Note that the determinism property guarantees that under every OR gate, for a given example $\mathbf{x}$, at most one wire has a flow of 1, and the rest is 0.

We are now ready to define the logistic circuit semantics.

Definition 2 (Logistic Circuit Semantics)**.** *A logistic circuit node n defines the following weight function $g_n(\mathbf{x})$.*

– *If n is a leaf (input) node, then $g_n(\mathbf{x}) = 0$.*

– *If n is an AND gate with p as its prime and s as its sub, then $g_n(\mathbf{x}) = g_p(\mathbf{x}) + g_s(\mathbf{x})$.*

– *If n is an OR gate with (child node, wire parameter) inputs $(c_1, \theta_1), \ldots, (c_m, \theta_m)$, then*

$$g_n(\mathbf{x}) = \sum_i f(n, \mathbf{x}, c_i) \cdot (g_{c_i}(\mathbf{x}) + \theta_i) \, .$$

Moreover, the entire logistic circuit α rooted in node n has weight $g_\alpha(\mathbf{x}) = g_n(\mathbf{x}) + \theta_b$, where θ_b denotes the bias parameter at the output. From this weight, we obtain the posterior distribution on class variable Y as

$$\Pr_\alpha(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-g_\alpha(\mathbf{x}))}. \quad (1)$$

With Boolean circuit flow, this definition essentially collects all the parameters on wires with flow 1 that reach the root to make a prediction. This is illustrated in Figure 1a by coloring red the gates and wires whose parameters and weight function are propagated upward. The logistic circuit in Figure 1a defines the same posterior predictions as the table in Figure 1b. Specifically, for the example assignment $A = 0$, $B = 1$, $C = 1$, $D = 0$, the weight function simply sums the parameters colored in red: $-4.6 - 1.2 + 2.3 + 3.9 + 1.5 = 1.9$. We then apply the logistic function (Eq. 1) to get the classification probability $\Pr(Y = 1 \mid \mathbf{x}) = \frac{1}{1+\exp(-1.9)} = 86.99\%$.

Real-Valued Data The semantics given so far assume Boolean inputs $\mathbf{x}$, which is a rather restrictive assumption and prohibits many machine learning applications. Next, we augment the logistic circuit semantics such that they can classify examples with continuous values.

Instead of having either 1 or 0, we now have real-valued variables $q \in [0, 1]$. We interpret each such variable as parameterizing an (independent) Bernoulli distribution (cf. [13]). Each continuous variable represents the probability of the corresponding Boolean random variable

X. For example, with $\mathbf{q}$ setting $A = 0.4$, $B = 0.8$, $C = 0.2$, and $D = 0.7$, the probability of $\neg A \wedge D$ would be $(1 - 0.4) \cdot 0.7 = 0.42$. The same distribution defines a probability for each logical sentence, and therefore each node in the logistic circuit. This allows us to generalize the notion of Boolean flow as follows.

Definition 3 (Continuous Flow). *Consider a deterministic OR gate n. Let $\mathbf{q}$ be a vector of probabilities, one for each variable in $\mathbf{X}$. The continuous flow $f(n, \mathbf{q}, c)$ of vector $\mathbf{q}$ between parent n and child c is*

$$f(n, \mathbf{q}, c) = \Pr_{\mathbf{q}}(c \mid n) = \frac{\Pr_{\mathbf{q}}(c)}{\Pr_{\mathbf{q}}(c \wedge n)} = \frac{\Pr_{\mathbf{q}}(c)}{\Pr_{\mathbf{q}}(n)},$$

where $\Pr_{\mathbf{q}}(.)$ is the fully-factorized distribution where each variable in $\mathbf{X}$ has the probability assigned by $\mathbf{q}$.

Logistic circuit semantics now support continuous data (after normalizing to $[0, 1]$), simply by replacing Boolean flow with continuous flow in Definition 2. Appendix B explains how the required probabilities can be computed efficiently, linear in the size of the logistic circuit.

3 Parameter Learning

In this section, we present how the parameters of a logistic circuit can be effectively learned from data.

Proposition 1. *Any logistic circuit model can be reduced to a logistic regression model over a particular feature set.*

Corollary 2. *Logistic circuit cross-entropy loss is convex.*

In order to reduce a logistic circuit to a logistic regression model, we need to write Equation 1 in the form

$$\frac{1}{1 + \exp(-\mathbb{x} \cdot \boldsymbol{\theta})},$$

where $\mathbb{x}$ is some vector of features extracted from the raw example $\mathbf{x}$ (or $\mathbf{p}$). This feature vector can only depend on $\mathbf{x}$ (or $\mathbf{p}$); not on the parameters $\boldsymbol{\theta}$. We show this transformation in Appendix A, together with efficient algorithms to calculate features $\mathbb{x}$ from examples in Appendix B.

Given this correspondence, any convex optimization technique can now be brought to bear on the problem of learning the parameters of a logistic circuit. In particular, we will use stochastic gradient descent for this task.

4 Structure Learning

This section presents an algorithm to learn a compact logical circuit structure for logistic circuits from data.

Learning Primitive The split operation was first introduced to modify the structure of PSDD circuits [6]. We adopt it here with minor changes[2] as the primitive operation for our structure learning algorithm. Splitting an AND gate happens by imposing two additional constraints that are *mutually exclusive* and *exhaustive*, in particular by making two opposing variable assignments. Executing a split creates partial copies of the gate and some of its decedents. Furthermore, one can choose to duplicate additional nodes up to a fixed depth (3 in our experiments). Appendix C gives further details of the split operation.

[2]Compared to the splits in LearnPSDD [6], we no longer require the constraints to only be on prime variables.

Table 1: Classification accuracy of logistic circuits in context with commonly used existing models.

ACCURACY % ON DATASET	MNIST	FASHION
BASELINE: LOGISTIC REGRESSION	85.3	79.3
BASELINE: KERNEL LOGISTIC REGRESSION[a]	97.7	88.3
3-LAYER MLP[b]	97.5	84.8
RAT-SPN [8]	98.1	89.5
5-LAYER MLP [c]	99.3	89.8
LOGISTIC CIRCUIT (BINARY)	97.4	87.6
LOGISTIC CIRCUIT (REAL-VALUED)	99.4	91.3
CNN WITH 3 CONV LAYERS [d]	99.1	90.7
RESNET [4]	99.5	93.6

[a]Based on the implementation of pixel n-grams from vowpal wabbit.
[b]Layers are of size 784-1000-500-250-10 respectively.
[c]Layers are of size 784-1000-500-250-2000-250-10 respectively.
[d]3-by-3 padded filters are used in convolutional layers.

Table 2: Number of parameters of logistic circuits in context with existing models, when achieving the classification accuracy reported in Table 1.

NUMBER OF PARAMETERS	MNIST	FASHION
BASELINE: LOGISTIC REGRESSION	<1K	<1K
BASELINE: KERNEL LOGISTIC REGRESSION	1,521 K	3,930K
LOGISTIC CIRCUIT (REAL-VALUED)	182K	467K
LOGISTIC CIRCUIT (BINARY)	268K	614K
3-LAYER MLP	1,411K	1,411K
RAT-SPN [8]	8,500K	650K
CNN WITH 3 CONV LAYERS	2,196K	2,196K
5-LAYER MLP	2,411K	2,411K
RESNET [4]	4,838K	4,838K

Learning Algorithm Next, we present the overall structure learning algorithm for logistic circuits, built on top of the split operation. Iteratively, one spit is executed to change the structure, followed by parameter learning. We only consider single-variable constraints. Our algorithm first selects which AND gate to split on, followed by a selection of which variable to execute a split with.

When using gradient descent optimization, one hopes the parameter on the AND gate output consistently has its partial derivatives point in the same direction for all training examples. This will steadily push the parameter to a large magnitude. If this is not the case, we will use splits to

Table 3: Comparison of logistic circuits with MLPs when trained with different percentages of the dataset.

ACCURACY % WITH % OF TRAINING DATA	MNIST			FASHION		
	100%	10%	2%	100%	10%	2%
5-LAYER MLP	99.3	**98.2**	94.3	89.8	86.5	80.9
CNN WITH 3 CONV LAYERS	99.1	98.1	95.3	90.7	87.6	83.8
LOGISTIC CIRCUIT (BINARY)	97.4	96.9	94.1	87.6	86.7	83.2
LOGISTIC CIRCUIT (REAL-VALUED)	**99.4**	97.6	**96.1**	**91.3**	**87.8**	**86.0**

alter the flow of examples through the circuit. Specifically, those AND gates whose associated output parameter has a large variance of its partial derivative (that is, the derivative of the loss function w.r.t. that parameter) requires splitting for the parameters to improve. We simply select the one with the highest training variance.

Given an AND gate to split, we consider candidate variables X to execute the split with. We construct two sets of training examples that affect this node: in one group, each example is weighted by the marginal probability of X; in the other, with the marginal probability of $\neg X$. Next, we calculate the within-group weighted variances of the partial derivatives. The variable with the smallest weighted variances gets picked, as this suggests the split will introduce new parameters with gradients that align.

5 Empirical Evaluation

In this section, we empirically evaluate the competitiveness of our learner on three aspects: classification accuracy, model complexity, and data efficiency.

Setup & Data Preprocessing We choose MNIST and Fashion[3] as our testbeds. Since logistic circuits are intended for binary classification, we use the standard "one vs. rest" approach to construct an ensemble multi-class classifier such that our method can be evaluated on these two datasets. When running the binary logistic circuit, we transform pixels that are smaller than their mean plus 0.05 standard deviation to 0 and the rest to 1. When running the real-valued version, we transform pixels to $[0, 1]$ by dividing them by 255. All experiments start with a predefined initial structure; we defer its details to Appendix D. The learned structure with the highest F1 score on validation after 48 hours of running is used for evaluation.

Classification Accuracy Table 1 summarizes the classification accuracy on test data. Learning a logistic circuit on the binary data is on par with a 3-layer MLP; the real-valued version outperforms 5-layer MLPs and even CNNs with 3 convolutional layers. The fact that logistic circuits achieve better accuracy than CNNs is surprising, since the logistic circuits do not use convolutions, which are specifically designed to exploit image invariances.

[3]A dataset consisting of Zalando's images. It is intended as a more challenging drop-in replacement of MNIST [12].

Besides, we would like to emphasis our comparison with the two baselines. As parameter learning of logistic circuits is equivalent with logistic regression, one can view structure learning of logistic circuits as a process of constructing composite features from raw samples. The significant improvement over standard logistic regression demonstrates the effectiveness of our method in extracting valuable features; using kernel logistic regression can only partially bridge the gap in performance, yet as shown later, it does so at the cost of many more parameters.

Furthermore, we want to call attention to our comparison with RAT-SPN. SPN is another form of circuit representation, with less restrictive structure. As so, parameter learning in SPN is not convex and generally requires other techniques such as EM. The empirical observation that our method achieves better classification accuracy than RAT-SPN demonstrates that in structure learning, imposing more restrictions on the model's structural syntax may be beneficial. The syntactical restriction of logistic circuits requires decomposability and determinism; without them, convex parameter learning does not appear to be possible. As structure learning is built on top of parameter learning, a well-behaved parameter-learning loss with a unique optimum can provide more informative guidance about how to adapt the structure, leading to a more competitive structure learning algorithm overall.

Model Complexity & Data Efficiency Table 2 summarizes the size of all compared models when achieving the reported accuracy. We can conclude that logistic circuits are significantly smaller than the alternatives. Table 3 summarizes the performance of the same compared models when only limited training samples are provided. Except on MNIST with 10% training samples, real-valued logistic circuits achieve the best classification accuracy. We refer to a more detailed discussion in Appendix E.

Acknowledgments

This work is supported by NSF grants #IIS-1657613, #IIS-1633857 and DARPA XAI grant #N66001-17-2-4032.

References

[1] J. Bekker, J. Davis, A. Choi, A. Darwiche, and G. Van den Broeck. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems 28 (NIPS)*, Dec. 2015.

[2] R. Benenson. What is the class of this image? `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`, 2018.

[3] A. Darwiche and P. Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

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

[5] D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

[6] Y. Liang, J. Bekker, and G. V. den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *UAI*, 2017.

[7] D. Lowd and P. Domingos. Learning arithmetic circuits. In *UAI*, pages 383–392, 2008.

[8] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, K. Kersting, and Z. Ghahramani. Probabilistic Deep Learning using Random Sum-Product Networks. *ArXiv e-prints*, June 2018.

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

[10] T. Rahman, P. Kothalkar, and V. Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 630–645. Springer, 2014.

[11] J. D. M. Rennie. Regularized logistic regression is strictly convex. Technical report, MIT, 2005.

[12] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[13] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. V. den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018.

A Proof of Proposition 1

The real question of transforming a logistic circuit's cross entropy loss to that of a logistic regression boils down to whether we can decompose g_n into $\mathbb{x} \cdot \theta$, which can be proven to be true by induction.

Proof.

– Base case: n is a leaf (input) node. It is obvious g_n can be expressed as $\mathbb{x} \cdot \theta$, as g_n always equals 0.

– Induction step: assume g of all the nodes under node n can be expressed as $\mathbb{x} \cdot \theta$. Then we need to consider two cases: (1) n is a logical AND gate; (2) n is a decision node (i.e. logical OR gate).

  * n is an AND gate with prime p and sub s. Given $g_p = \mathbb{x}_p \cdot \theta_p$ and $g_s = \mathbb{x}_s \cdot \theta_s$,

$$\begin{aligned} g_n &= \mathbb{x}_p \cdot \theta_p + \mathbb{x}_s \cdot \theta_s \\ &= \begin{bmatrix} \mathbb{x}_p \\ \mathbb{x}_s \end{bmatrix} \cdot \begin{bmatrix} \theta_p \\ \theta_s \end{bmatrix} \end{aligned}$$

  * n is an OR gate with (child node, wire parameter) inputs $\{(e_1, \theta_1), \dots, (e_m, \theta_m)\}$. Given $g_{e_i} = \mathbb{x}_{e_i} \cdot \theta_{e_i}$,

$$\begin{aligned} g_n &= \sum_i f(n, \mathbf{x}, e_i) \cdot (\mathbb{x}_{e_i} \cdot \theta_{e_i} + \theta_i) \\ &= \begin{bmatrix} f(n, \mathbf{x}, e_1) \cdot \mathbb{x}_{e_1} \\ f(n, \mathbf{x}, e_1) \\ \dots \\ f(n, \mathbf{x}, e_m) \cdot \mathbb{x}_{e_m} \\ f(n, \mathbf{x}, e_m) \mid \mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \theta_{e_1} \\ \theta_1 \\ \dots \\ \theta_{e_m} \\ \theta_m \end{bmatrix} \end{aligned}$$

□

Note this proof holds valid regardless of whether the input sample $\mathbf{x}$ is binary or real-valued. With this proof, it is obvious that learning parameters of a logistic circuit is equivalent to optimizing a logistic regression. This is also the main motivation why we name our representation as logistic circuit. Due to the space limit, we do not present the detailed proof that logistic regression is convex; we refer readers to [11].

B Calculation of Features

In this section, we present how features can be efficiently calculated from samples. Note in our algorithms we calculate features for all nodes (OR gates, AND gates and leaves). However, when calculating the classification

Algorithm 1: Feature vector $\mathbb{x}$ from binary sample $\mathbf{x}$.

```
Input: an example $\mathbf{x}$ from the training data set
Result: feature vector $\mathbb{x}$ with entries 0 or 1.
$\mathbb{x}$(root) := CalculateFeature(root)
Function CalculateFeature($n$):
    if IsLeaf($n$) then
        $\mathbb{x}(n) := \mathbf{x}(n.variable) \models [n]$
    else if IsAndGate($n$) then
        $\mathbb{x}(n) :=$
            CalculateFeature($n.prime$)
              $\cdot$ CalculateFeature($n.sub$)
    else
        // $n$ is an OR gate
        $\mathbb{x}(n) := 0$
        for $e$ in $n$.inputs do
            if CalculateFeature($e$) = 1 then
                $\mathbb{x}(n) := 1$
                break
    return $\mathbb{x}(n)$
```

Algorithm 3: Feature $\mathbb{x}$ from real-valued sample $\mathbf{x}$.

```
Input: An example $\mathbf{x}$ from the training data set
Result: Feature vector $\mathbb{x}$, with entries from [0,1].
for $n$ in all non-leaf nodes parents before children do
    $\mathbb{x}(n) := 0$
$\mathbb{x}$(root) := $p$([root] | $\mathbf{x}$)
for $n$ in all non-root OR gates parents before children do
    for $e$ in $n$.inputs do
        $\mathbb{x}(e) := \mathbb{x}(n) \cdot p([e] \mid \mathbf{x}) \;/\; p([n] \mid \mathbf{x})$
        $\mathbb{x}(e.prime) += \mathbb{x}(e)$
        $\mathbb{x}(e.sub) += \mathbb{x}(e)$
```

Algorithm 2: Node probability of all nodes given $\mathbf{x}$, where $\mathbf{x}$ is real-valued. This is a bottom-up pass.

```
Input: an example $\mathbf{x}$ from the training data set
Result: $p([n] \mid \mathbf{x})$: the node probability of $[n]$ given $\mathbf{x}$.
        $p(\cdot)$ will be used in the top-down pass
        (Algorithm 3) to obtain the final feature vector.
for $n$ in the circuit's nodes children before parents do
    if IsLeaf($n$) then
        $X := n$.variable
        if $[n]$ is $X$ then
            $p([n] \mid \mathbf{x}) = \mathbf{x}(X)$
        else
            // $[n]$ is $\neg X$
            $p([n] \mid \mathbf{x}) = 1 - \mathbf{x}(X)$
    else if IsAndGate($n$) then
        $p([n] \mid \mathbf{x}) = p([n.prime] \mid \mathbf{x}) \cdot p([n.sub] \mid \mathbf{x})$
    else
        // $n$ is an OR gate
        $p([n] \mid \mathbf{x}) := 0$
        for $e$ in $n$.inputs do
            $p([n] \mid \mathbf{x}) += p([e] \mid \mathbf{x})$
```

Algorithm 4: Split($e, n, \mathbf{c_1}, \mathbf{c_2}, d$)

```
Input: $e$: the original AND gate to be split.
       $n$: the AND gate's parent OR gate.
       $\mathbf{c}$ : mutually exclusive and exhaustive constraints.
       $d$: depth of PartialCopy.
Result: $e$ is split by constraining on $\mathbf{c_1}, \mathbf{c_2}$ .
RemoveAndGate($n, e$)
if ContainVariables($e$.prime, $\mathbf{C}$) then
    $p_1$ := PartialCopy($e$.prime, $\mathbf{c_1}$, $d$)
    $p_2$ := PartialCopy($e$.prime, $\mathbf{c_2}$, $d$)
    $s_1$ := PartialCopy($e$.sub, true, $d$)
    $s_2$ := PartialCopy($e$.sub, true, $d$)
else
    $p_1$ := PartialCopy($e$.prime, true, $d$)
    $p_2$ := PartialCopy($e$.prime, true, $d$)
    $s_1$ := PartialCopy($e$.sub, $\mathbf{c_1}$, $d$)
    $s_2$ := PartialCopy($e$.sub, $\mathbf{c_2}$, $d$)
AndAndGate($n$, NewAndGate($p_1, s_1$))
AddAddGate($n$, NewAndGate($p_2, s_2$))
```

Algorithm 5: PartialCopy$(n, \mathbf{c}, d)$

```
Input: $n$: node to copy;  $\mathbf{c}$ : constraint, which is an
       assignment to one variable in our setting;
       $d$: depth of PartialCopy.
Result: constrained copy of $n$
if IsLeaf($n$) then
    if $\mathbf{c}$.variable = $n$.variable then
        if $\mathbf{c} \models [n]$ then
            return $n$
        else if $[n] = \top$ then
            return NewLeaf($\mathbf{c}$)
        else
            return None
    else
        return $n$
else if IsAndGate($n$) then
    if $d > 0$ then
        $p$ := PartialCopy($n$.prime, $\mathbf{c}$, $d-1$)
        $s$ := PartialCopy($n$.sub, $\mathbf{c}$, $d-1$)
    else
        if $\mathbf{c} \models [n.prime]$ then
            $p$ := PartialCopy($n$.prime, $\mathbf{c}$, 0)
        else
            $p := n$.prime
        if $\mathbf{c} \models [n.sub]$ then
            $s$ := PartialCopy($n$.sub, $\mathbf{c}$, 0)
        else
            $s := n$.sub
    return NewAndGate($p$, $s$)
else
    // $n$ is an OR gate
    $E := \varnothing$
    for $e$ in $n$.inputs do
        $e'$ :=PartiCopy($e$, $\mathbf{c}$, d)
        $E := E \cup e'$
    return NewOrGate($E$)
```

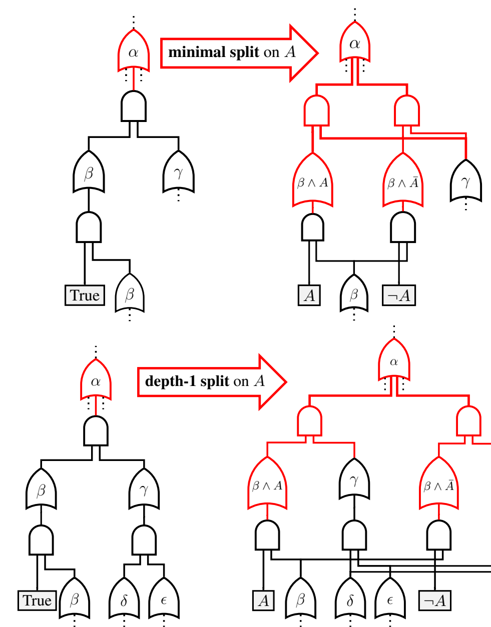


Figure 2: Split. Decision nodes have their base labeled.

probability, only features associated with AND gates are used (see Definition 2 and Figure 1)[4].

B.1 Binary Dataset

When samples are binary, we can take advantage of the deterministic property, leading to an algorithm with average time complexity of $O(\log N)$[5], where N denotes the number of nodes in the logistic circuit; see Algorithm 1. Note these nodes that do not get visited would have feature of 0, and thus can be omitted in the later calculation of classification probability.

B.2 Real-Valued Dataset

When input $\mathbf{x}$ is real-valued, the deterministic property is relaxed and two passes over the whole logistic circuit (one bottom-up and one top-down) are necessary, resulting in an $O(N)$ method; see Algorithm 2 and 3 for details.

Note instead of inputing one single sample each time, one can directly supply Algorithm 2 and 3 with a vector of samples. Our proposed calculation method is completely compatible with matrix operations, and by doing so, one can expect a large speedup.

C Details of Split Operation

We require the partial assignments (i.e. the newly imposed constraints) in splits do not overlap both the prime and sub

[4]In feature calculation, AND gates actually act on behalf of the wires from their parent OR gates to them.

[5]In worst case, time complexity is $O(N)$.

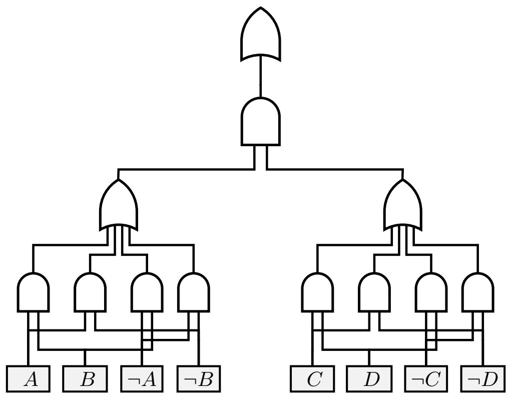


Figure 3: Initial structure of logistic circuits with 4 pixels (8 terminal features).

variables. The mutual exclusiveness and exhaustiveness of constraints ensure the original AND gate's parent OR gate remains deterministic after the split. To execute a split, the original AND gate is removed and a new AND gate is created for every partial assignment; see Algorithm 4 for details.

PartialCopy make a copy and its descents up to the specified level d. The AND gates of the copy beyond the specified level are redirected to the original logistic circuit. One can supply PartialCopy with a constraint. In that case, only descendants that agree with the constraint are kept. In order to propagate the constraint properly, descents beyond depth d may be altered. See Algorithm 5 for details.

A minimal split has PartialCopies with $d = 0$ and thus only expand as few elements and decision nodes as possible. Any non-minimal operations ($d \neq 0$) can be reproduced by executing multiple minimal operations. The difference between a minimal split and a depth-1 split is visualized in Figure 2.

D Initial Structure

All experiments in this paper start with an initial structure where every pixel has two corresponding features, one for the pixel being true and the other false. Pixels are paired up by AND gates at the level immediately above the terminal. To be more specific, an AND gate is created for every joint assignment to the pair. AND gates that are created for the same pair would share one OR gate parent. After this, OR gates are paired with AND gates and every AND gate is connected to its own OR gate parent until we reach the root. Figure 3 is an example of the initial structure when there are 4 pixels. Note our proposed structure learning algorithm is fully compatible with other initial structures and one can create different ad hoc initializations tailored to the specific applications.

E Data Efficiency

From a top-down perspective, each OR gate of a logistic circuit presents a weighted choice between its wires. Given so, one can view a logistic circuit as a decision diagram. Under this perspective, splits refine OR gates' branching rules. As each branching rule naturally applies to multiple samples, we suspect that the splits selected by our structure learning algorithm can reflect the very general conditional feature information in the dataset and thus may be very data efficient.

We design experiments to specifically investigate how well our structure learning algorithm performs under the setting where the number of training samples is limited. We have two additional sets of experiments, where only 2% and 10% of the original training data is supplied respectively. We mainly compare against 5-layer MLP and CNN with 3 convolutional layers, whose performance is on par with our method. As summarized in Table 3, except on MNIST with 10% training samples, real-valued logistic circuits achieve the best classification accuracy. Moreover, in both versions of logistic circuits, when the available training samples are reduced from 100% to 2%, the accuracy only drops by around 3% when evaluating on MNIST; around 5% on Fashion. In contrast, a much larger drop occurs for 5-layer MLP and CNN. To be more specific, MLP's accuracy drops by 5% (9%) while CNN's accuracy drops by 4% (7%) on MNIST (Fashion). This small magnitude of accuracy decrease illustrates how data efficient the proposed structure learning algorithm is.