Machine Learning (2023) 112:2821–2843
https://doi.org/10.1007/s10994-023-06346-5

Differentiable learning of matricized DNFs and its application to Boolean networks

Taisuke Sato[1] · **Katsumi Inoue**[1]

Received: 6 June 2022 / Revised: 4 April 2023 / Accepted: 8 May 2023 /
Published online: 21 June 2023

Abstract
Boolean networks (BNs) are well-studied models of genomic regulation in biology where nodes are genes and their state transition is controlled by Boolean functions. We propose to learn Boolean functions as Boolean formulas in disjunctive normal form (DNFs) by an explainable neural network Mat_DNF and apply it to learning BNs. Directly expressing DNFs as a pair of binary matrices, we learn them using a single layer NN by minimizing a logically inspired non-negative cost function to zero. As a result, every parameter in the network has a clear meaning of representing a conjunction or literal in the learned DNF. Also we can prove that learning DNFs by the proposed approach is equivalent to inferring interpolants in logic between the positive and negative data. We applied our approach to learning three literature-curated BNs and confirmed its effectiveness. We also examine how generalization occurs when learning data is scarce. In doing so, we introduce two new operations that can improve accuracy, or equivalently generalizability for scarce data. The first one is to append a noise vector to the input learning vector. The second one is to continue learning even after learning error becomes zero. The first one is explainable by the second one. These two operations help us choose a learnable DNF, i.e., a root of the cost function, to achieve high generalizability.

Keywords Boolean function · DNF · Boolean network · Neural network · Generalization

Editors: Alireza Tamaddoni-Nezhad, Alan Bundy, Luc De Raedt, Artur d'Avila Garcez, Sebastijan Dumančić, Cèsar Ferri, Pascal Hitzler, Nikos Katzouris, Denis Mareschal, Stephen Muggleton, Ute Schmid.

✉ Taisuke Sato
taisukest@gmail.com

Katsumi Inoue
inoue@nii.ac.jp

[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

1 Introduction

Boolean networks (BNs) are a simple yet effective model of gene regulatory networks where nodes are genes and their state transition is controlled by Boolean functions (Kauffman, 1969). They have been studied mathematically (Cheng and Qi, 2010; Kobayashi and Hiraishi, 2014), logically in AI (Inoue et al., 2014; Tourret et al., 2017; Chevalier et al., 2019; Gao et al., 2022) and from the viewpoint of deep learning (Zhang et al., 2019). Their learning is reduced to learning Boolean functions from a set of input–output pairs and can be carried out for example by the REVEAL algorithm (Liang et al., 1998) or by the BestFit extension algorithm (Lähdesmäki et al., 2003).

In this paper, we propose a new approach to learning Boolean functions. We introduce a simple ReLU neural network (NN) called Mat_DNF that learns Boolean functions and outputs Boolean formulas in disjunctive normal form (DNFs). We represent a DNF by a pair $(\mathbf{C}, \mathbf{D})$ of binary matrices where $\mathbf{C}$ stands for conjunctions and $\mathbf{D}$ a disjunction respectively. Mat_DNF learns a matricized DNF $(\mathbf{C}, \mathbf{D})$ as network parameters from the learning data by minimizing a non-negative cost function $\mathrm{J}(\mathbf{C}, \mathbf{D})$ to zero. As a result, every network parameter in Mat_DNF has a clear meaning of (potentially) denoting a literal or a conjunction (disjunct[1]) in the learned DNF.

Although there exist several ways to represent Boolean functions such as decision trees (Oliveira and Sangiovanni-Vincentelli, 1993), polynomial threshold functions (Hansen and Podolskii, 2015), Boolean circuits (Malach and Shalev-Shwartz, 2019) and support vector machines (Mixon and Peterson, 2015), we choose DNFs for two reasons: one is explainability and the other is to relate the learning process to logical inference. Explainability is guaranteed as our network parameters directly represent a matricized DNF. Moreover since the learned output is a DNF, exploring the logical relationship between the learning data and the learned DNF becomes possible and we find that the learned DNF is what is called an interpolant in logic (Craig, 1957) interpolating between the positive and negative input data, which uncovers a new connection that connects neural learning to symbolic inference.

Boolean function learning can be either discrete or continuous. One group such as SAT encoding with integer programming (Kamath et al., 1992) and stochastic local search (Ruckert and Kramer, 2003) works in discrete spaces. The other group uses NNs in continuous spaces such as simulating Boolean circuits (Malach and Shalev-Shwartz, 2019), Neural Logic Networks (Payani and Fekri, 2019) and Net-DNF (Katzir et al., 2021). Our learning is just between the two. Unlike the former, Mat_DNF is differentiable[2]. Unlike the latter, it explicitly operates on matricized DNFs, discrete expressions, which are not implicitly embedded in the neural network architecture.

In the context of BN learning, Mat_DNF offers a robust yet explainable end-to-end approach as an alternative to previous ones (Liang et al., 1998; Lähdesmäki et al., 2003;

[1] For a disjunction $A \vee B$, the subformula A (resp. B) is called a disjunct. If $A \vee B$ is a DNF, each disjunct is a conjunction of literals. Disjuncts are sometimes called terms.

[2] Mat_DNF uses $\min_1(x) = \min(x, 1)$ in stead of $\mathrm{ReLU}(x)$ (note $\min_1(x) = 1 - \mathrm{ReLU}(1 - x)$ holds) for computing disjucntion, which is originated from real-valued Łukasiewicz logic. $\min_1(x)$ is non-differentiable at $x = 1$. However non-differentiable points form a Lebesgue measure zero set and the probability of hitting a non-differentiable point in learning is zero. So practically the non-differentiability of $\min_1(x)$ causes no problem. Theoretically the subgradient method may be usable in a special case. There are other types of differentiable logic and logical operators applicable to them (van Krieken et al., 2022).

Inoue et al., 2014; Tourret et al., 2017; Gao et al., 2022). Compared to the REVEAL algorithm (Liang et al., 1998) and the BestFit extension algorithm (Lähdesmäki et al., 2003), Mat_DNF imposes no limit on the number of function variables. So if there are 18 genes (Irons, 2009), DNFs in 18 variables are considered. The LF1T algorithm (Inoue et al., 2014) symbolically learns a BN represented as a ground normal logic program from state transitions. Generalization is done by resolution. The NN-LFIT algorithm (Tourret et al., 2017) adopts a two-stage approach that learns features by a feed-forward NN and extracts DNFs from the learned parameters. D-LFIT (Gao et al., 2022) takes a further elaborated approach of combining two neural networks to reduce search space. By comparison, Mat_DNF is a much simpler single layer NN whose learned parameters directly represent a DNF and there is no need for post processing.

To improve the accuracy of the DNF learned from insufficient data, we introduce two operations. The first one is "noise-expansion". It appends a noise vector to the input learning vector.[3] The second one is "over-iteration" which keeps learning even after learning error becomes zero. Since adding a noise vector causes extra steps of parameter update while moving around local minima of the cost function J, the net effect of the first one is attributable to the second one. The fact that these two operations can considerably improve accuracy means that the choice of a root of the cost function $J(\mathbf{C}, \mathbf{D}) = 0$, or more generally the choice of a local minimum significantly affects accuracy and generalizability.

Finally we confirm the effectiveness of our approach through three learning experiments with literature-curated BNs (Fauré et al., 2006; Irons, 2009; Krumsiek et al., 2011). We applied Mat_DNF to learning data generated from these BNs to see if Mat_DNF can recover the original DNFs in BNs. For the first two synchronous BNs (Fauré et al., 2006; Irons, 2009), the recovery rate is high. By detailed analysis of the learning results, it is suggested that this high recovery rate is due to the effect of over-iteration caused by implicit noise-expansion. However, the third asynchronous BN (Krumsiek et al., 2011; Ribeiro et al., 2021) presents a much more difficult case and only six DNFs are completely recovered out of 11 original DNFs, though this result is comparable to that of rfBFE (Gao et al., 2018), one of the state-of-the-art BN learning algorithms.

Thus our contributions are three fold. First a proposal of new approach to the end-to-end learning of Boolean functions by an explainable single layer NN Mat_DNF together with its application to BN learning, second the establishment of the equivalence between neural learning of DNFs by Mat_DNF and symbolic inference of DNFs as interpolants between the positive and negative data and third the introduction of two new operations, noise-expansion and over-iteration, that can improve accuracy by shifting the choice of a local minimum.

In what follows, after a preliminary section, we introduce Mat_DNF in Sect. 3. We then prove the relationship between the learning by Mat_DNF and the inference of interpolants in logic in Sect. 4. Section 5 examines the behavior of Mat_DNF w.r.t. insufficient learning data and introduces noise-expansion and over-iteration that improve accuracy. Section 6 reports three BN learning experiments and Sect. 7 discusses related work. Section 8 is the conclusion.

[3] In this paper, noise does not mean classification noise but irrelevant bits in the learning data that disturb Boolean function learning.

2 Preliminaries

Throughout this paper, bold italic capital letters such as $\mathbf{A}$ stand for matrices and so do bold italic lower case letters such as $\mathbf{a}$ for vectors. We equate a one-dimensional matrix with a vector. The i-th element of $\mathbf{a}$ is designated by $\mathbf{a}(i)$ and the i, j-th element of $\mathbf{A}$ by $\mathbf{A}(i, j)$. Given two $m \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$, $[\mathbf{A};\mathbf{B}]$ represents the $2\,m \times n$ matrix of $\mathbf{A}$ stacked onto $\mathbf{B}$. $\|\mathbf{a}\|_1 = \sum_i \mid \mathbf{a}(i) \mid$ denotes the 1-norm of $\mathbf{a}$ and $\|\mathbf{A}\|_F$ the Frobenius norm of $\mathbf{A}$. Let $\mathbf{a}$ and $\mathbf{b}$ be n dimensional vectors. Then $(\mathbf{a} \bullet \mathbf{b})$ stands for their inner product (dot product) and $\mathbf{a} \odot \mathbf{b}$ their Hadamard product, i.e., $(\mathbf{a} \odot \mathbf{b})(i) = \mathbf{a}(i)\mathbf{b}(i)$ for $i(1 \leq i \leq n)$. For a scalar θ, $(\mathbf{a})_{\geq\theta}$ denotes a binary vector such that $(\mathbf{a})_{\geq\theta}(i) = 1$ if $\mathbf{a}(i) \geq \theta$ and $(\mathbf{a})_{\geq\theta}(i) = 0$ otherwise for $i(1 \leq i \leq n)$. Similarly $1 - \mathbf{a}$ denotes the complement of $\mathbf{a}$, i.e. $(1 - \mathbf{a})(i) = 1 - \mathbf{a}(i)$ for $i(1 \leq i \leq n)$. These notations naturally extend to matrices like $(\mathbf{A})_{\geq\theta}$ and $1 - \mathbf{A}$. $\min_1(x) = \min(x, 1)$ is a function returning the lesser of 1 and x. $\min_1(\mathbf{A})$ is the component-wise application of $\min_1(x)$ to $\mathbf{A}$. We implicitly assume that all dimensions of vectors and matrices in various expressions are compatible. Let $d_1 \vee \cdots \vee d_h$ be a DNF in n variables. If every disjunct d_i is a conjunction of n distinct literals, it is said to be full. For a set S, $\mid S \mid$ stands for the number of elements in S.

3 Learning DNFs in vector spaces

3.1 Evaluating matricized DNFs

Let $\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge \neg x_3)$ be a DNF in three variables. φ has two disjuncts $(x_1 \wedge x_2)$ and $(x_1 \wedge \neg x_3)$. We represent φ by a pair $(\mathbf{C}, \mathbf{D})$ of binary matrices:

$$\begin{array}{c} \quad x_1 \;\; x_2 \;\; x_3 \;\; \neg x_1 \;\; \neg x_2 \;\; \neg x_3 \\ \mathbf{C} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \mathbf{D} = [1 \;\; 1]$$

As can be seen, each row of $\mathbf{C}$ represents a disjunct (conjunction of literals) of φ. For example, the first row of $\mathbf{C}$ represents the first disjunct $(x_1 \wedge x_2)$ by setting $\mathbf{C}(1, 1) = \mathbf{C}(1, 2) = 1$. $\mathbf{D}$ on the other hands represents the choice of a conjunction as a disjunct; in the current case, both disjuncts in $\mathbf{C}$ are chosen as disjunct of φ as designated by $\mathbf{D} = [1\ 1]$. If $\mathbf{D} = [1\ 0]$, φ will contain only the first disjunct $(x_1 \wedge x_2)$ in $\mathbf{C}$. Generally a DNF φ in n variables with at most h disjuncts is represented by an $h \times 2n$ binary matrix $\mathbf{C}$ and a $1 \times h$ binary matrix $\mathbf{D}$. By default, we consider a DNF φ and its matrix representation $(\mathbf{C}, \mathbf{D})$ exchangeable and call $(\mathbf{C}, \mathbf{D})$ matricized DNF φ.

Now we describe how φ is evaluated as a Boolean function $\varphi(\mathbf{x})$ over its domain $\mathbf{I}_0 = \{1, 0\}^n$ of bit sequences. Each $\mathbf{x} \in \mathbf{I}_0$ is equated with a binary column vector called "interpretation vector" representing an interpretation (assignment) such that a variable x_j $(1 \leq j \leq n)$ is mapped to $\mathbf{x}(j) \in \{1, 0\}$. Henceforth for convenience we treat $\mathbf{I}_0$ as an $n \times 2^n$ binary matrix packed with such 2^n possible interpretation vectors and specifically call it the domain matrix for n variables.

Let $\mathbf{x}$ be an interpretation vector in $\mathbf{I}_0$. A matricized DNF $\varphi = (\mathbf{C}(h \times 2n), \mathbf{D}(1 \times h))$ is evaluated by $\mathbf{x}$ as follows. First compute a column vector $\mathbf{N} = \mathbf{C}[(1 - \mathbf{x});\mathbf{x}]$. $\mathbf{N}(j)\,(1 \leq j \leq h)$

denotes the number of literals contained in the j-th conjunction of $\mathbf{C}$ and falsified by $\mathbf{x}$, and hence $\min_1(\mathbf{N})(j) = 0$ holds if-and-only-if the j-th conjunction is false in $\mathbf{x}$. Next compute a column vector $\mathbf{M} = 1 - \min_1(\mathbf{N})$ which is the bit inversion of $\min_1(\mathbf{N})$ and $\mathbf{M}(j)$ gives the truth value $\in \{0, 1\}$ of the j-th conjunction in $\mathbf{C}$. Finally compute a scalar $\mathbf{V} = \mathbf{DM}$. It denotes the number of disjuncts in φ satisfied by $\mathbf{x}$. Hence $(\mathbf{V})_{\geq 1} \in \{0, 1\}$ gives the truth value of φ evaluated by $\mathbf{x}$. Write $\mathbf{x} \models \varphi$ when φ is true in $\mathbf{x}$, i.e. $\mathbf{x}$ satisfies φ. In fact we have $\mathbf{x} \models \varphi$ if-and-only-if $(\mathbf{V})_{\geq 1} = 1$.

Write $\mathbf{C} = [\mathbf{C}^P\ \mathbf{C}^N]$ where $\mathbf{C}^P(h \times n)$ (resp. $\mathbf{C}^N(h \times n)$) is a submatrix representing positive (resp. negative) occurrences of variables in φ. Then the whole evaluation process is described by one line (1):

$$\varphi(\mathbf{x}) = (\mathbf{D}(\mathbf{1}_h - \min_1(\mathbf{C}[(\mathbf{1}_n - \mathbf{x}); \mathbf{x}])))_{\geq 1} \tag{1}$$

$$\begin{aligned} &= (\mathbf{D}(\mathbf{1}_h - \min_1((\mathbf{C}^N - \mathbf{C}^P)\mathbf{x} + \mathbf{C}^P\mathbf{1}_n)))_{\geq 1} \\ &= (\mathbf{D}(\mathrm{ReLU}((\mathbf{C}^P - \mathbf{C}^N)\mathbf{x} + \mathbf{1}_h - \mathbf{C}^P\mathbf{1}_n)))_{\geq 1} \\ &\quad \text{because} \mathrm{ReLU}(x) = \max(x, 0) = 1 - \min_1(1 - x) \end{aligned} \tag{2}$$

where $\varphi(\mathbf{x})$ denotes the truth value $\in \{0, 1\}$ of φ as a Boolean function evaluated by $\mathbf{x}$. This notation is naturally extended to a set of interpretation vectors like $\varphi(\mathbf{I}_0)$. $\mathbf{1}_h$ and $\mathbf{1}_n$ are all-one vectors of length h and n respectively. We rewrite (1) to (2). What the latter tells us is that our evaluation process is exactly a forward pass of a single layer ReLU network consisting of a linear output layer and a hidden layer with a weight matrix $\mathbf{C}^P - \mathbf{C}^N$ and a bias vector $\mathbf{1}_h - \mathbf{C}^P\mathbf{1}_n$. We name this ReLU network Mat_DNF. It is a simple NN specialized for DNFs derived from the evaluation process of a DNF where the disjunction $x \vee y$ is replaced by $\min_1(x + y)$ as in Łukasiewicz's many valued logic.

3.2 Learning DNFs by Mat_DNF

By adding a backward pass to the equation (1), Mat_DNF can learn Boolean functions. Here we describe how Mat_DNF learns them. Let f be a target Boolean function in n variables and $\mathbf{I}_0 = [\mathbf{x}_1 \cdots \mathbf{x}_{2^n}]$ the domain matrix for n variables. In learning, we are given a submatrix $\mathbf{I}_1(n \times l) = [\mathbf{x}_{i_1} \cdots \mathbf{x}_{i_l}]$ $(l \leq 2^n)$ of $\mathbf{I}_0$. $\mathbf{I}_1$ is mapped by f to a $1 \times l$ row vector $\mathbf{I}_2 = f(\mathbf{I}_1) = [f(\mathbf{x}_{i_1}) \cdots f(\mathbf{x}_{i_l})]$. $(\mathbf{I}_1, \mathbf{I}_2) = (\mathbf{I}_1, f(\mathbf{I}_1))$ is called an input–output pair for f and $\mathbf{I}_1$ its input domain. Learning a DNF φ here thus means a learner receives an input–output pair $(\mathbf{I}_1, \mathbf{I}_2) = (\mathbf{I}_1, f(\mathbf{I}_1))$ for a target Boolean function f and returns a DNF φ such that $\varphi(\mathbf{I}_1) = \mathbf{I}_2$. Mat_DNF receives $(\mathbf{I}_1, \mathbf{I}_2)$ and returns a matricized DNF φ such that $\varphi(\mathbf{I}_1) = \mathbf{I}_2$ when it stops with learning error = 0.

Let $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ be real matrices. They are relaxation versions of $\mathbf{C}$ and $\mathbf{D}$. Introduce $\max_0(x) = \max(x, 0)$ (ReLU), $\widetilde{\mathbf{N}} = \widetilde{\mathbf{C}}[(1 - \mathbf{I}_1); \mathbf{I}_1]$, $\widetilde{\mathbf{M}} = 1 - \min_1(\widetilde{\mathbf{N}})$, $\widetilde{\mathbf{V}} = \widetilde{\mathbf{D}}\widetilde{\mathbf{M}}$, $\mathrm{Y} = \|\widetilde{\mathbf{C}} \odot (1 - \widetilde{\mathbf{C}})\|_F^2$ and $\mathrm{Z} = \|\widetilde{\mathbf{D}} \odot (1 - \widetilde{\mathbf{D}})\|_F^2$. Then define a non-negative cost function $\mathrm{J}(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}})$ by

$$\mathrm{J} = (\mathbf{I}_2 \bullet (1 - \min_1(\widetilde{\mathbf{V}}))) + ((1 - \mathbf{I}_2) \bullet \max_0(\widetilde{\mathbf{V}})) + (1/2)\mathrm{Y} + (1/2)\mathrm{Z}. \tag{3}$$

The first term $(\mathbf{I}_2 \bullet (1 - \min_1(\widetilde{\mathbf{V}})))$ is a non-negative scalar and deals with the case of $f(\mathbf{x}_{i_j}) = \mathbf{I}_2(i_j) = 1$ $(1 \le j \le l)$. Likewise the second term $((1 - \mathbf{I}_2) \bullet \max_0(\widetilde{\mathbf{V}}))$ is non-negative and takes care of the case of $f(\mathbf{x}_{i_j}) = \mathbf{I}_2(i_j) = 0$. Y and Z are penalty terms to make $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ binary respectively.

Proposition 1 *$J(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}) = 0$ if-and-only-if $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ are binary matrices representing a DNF φ such that $\varphi(\mathbf{I}_1) = \mathbf{I}_2$.*

Proof We prove only-if part. The converse is obvious. Suppose $\mathrm{J} = \mathrm{J}(\widetilde{\mathbf{C}},\widetilde{\mathbf{D}}) = 0$. Every term in (3) is zero. $\mathrm{Y} = \mathrm{Z} = 0$ immediately implies $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ are binary. Let φ be a DNF represented by them. The first term deals with the case of $\mathbf{I}_2(i_j) = f(\mathbf{x}_{i_j}) = 1$ $(1 \le j \le l)$. It is a sum of non-negative summands of the form $(1 - \min_1(\widetilde{\mathbf{V}}(i_j)))$. Hence $\mathrm{J} = 0$ implies $\min_1(\widetilde{\mathbf{V}}(i_j)) = 1$, i.e. φ is true in $\mathbf{x}_{i_j} \in \mathbf{I}_1$ when $\mathbf{I}_2(i_j) = 1$. The second term is dual to the first term, dealing with the case of $\mathbf{I}_2(i_j) = 0$. Similarly to the first term, we can prove that φ is false in $\mathbf{x}_{i_j} \in \mathbf{I}_1$ when $\mathbf{I}_2(i_j) = 0$. By combining the two, we conclude that φ gives $\mathbf{I}_2$ when evaluated by $\mathbf{I}_1$, i.e., $\varphi(\mathbf{I}_1) = \mathbf{I}_2$. □

Learning by Mat_DNF is carried out based on **Proposition** 1 by minimizing J until $\mathrm{J} = 0$ using gradient descent. $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ are iteratively updated by their Jacobians, $\mathbf{J}_a^{\tilde{C}}$ for $\widetilde{\mathbf{C}}$ and $\mathbf{J}_a^{\tilde{D}}$ for $\widetilde{\mathbf{D}}$, for example like $\widetilde{\mathbf{C}} = \widetilde{\mathbf{C}} - \alpha \mathbf{J}_a^{\tilde{C}}$ where $\alpha > 0$ is a learning rate. To compute the Jacobians, we introduce $\widetilde{\mathbf{W}} = -(\widetilde{\mathbf{V}})_{\le 1} \odot \mathbf{I}_2 + (\widetilde{\mathbf{V}})_{\ge 0} \odot (1 - \mathbf{I}_2)$. Then $\mathbf{J}_a^{\tilde{C}}$ and $\mathbf{J}_a^{\tilde{D}}$ are computed by (4).

$$\begin{aligned} \mathbf{J}_a^{\tilde{C}} &= ((-(\widetilde{\mathbf{N}})_{\le 1}) \odot (\widetilde{\mathbf{D}}^T \widetilde{\mathbf{W}}))[(1 - \mathbf{I}_1) \mathbf{I}_1]^T + (1 - 2\widetilde{\mathbf{C}}) \odot \mathbf{Y} \\ \mathbf{J}_a^{\tilde{D}} &= \widetilde{\mathbf{W}} \widetilde{\mathbf{M}}^T + (1 - 2\widetilde{\mathbf{D}}) \odot \mathbf{Z} \end{aligned} \tag{4}$$

These Jacobians are derived as follows. We first derive $\mathbf{J}_a^{\tilde{C}}$. Let $\widetilde{\mathbf{C}}_{pq} = \widetilde{\mathbf{C}}(p, q)$ be an arbitrary element of $\widetilde{\mathbf{C}}$. Put $\Delta_Y = (1 - 2\widetilde{\mathbf{C}}) \odot \mathbf{Y}$. We have

$$\begin{aligned} \partial \widetilde{\mathbf{M}} / \partial \widetilde{\mathbf{C}}_{pq} &= -\partial \min_1(\widetilde{\mathbf{N}}) / \partial \widetilde{\mathbf{C}}_{pq} \\ &= -(\widetilde{\mathbf{N}})_{\le 1} \odot (\mathbf{I}_{pq}(1 - [\mathbf{I}_1 (1 - \mathbf{I}_1)])) \end{aligned}$$

where $\mathbf{I}_{pq}$ is a zero matrix except for the p, q-th element which is 1. We use $(\mathbf{A} \bullet \mathbf{B}) = \sum_{i,j} \mathbf{A}(i,j)\mathbf{B}(i,j)$ to denote the dot product of $\mathbf{A}$ and $\mathbf{B}$. Note $(\mathbf{A} \bullet (\mathbf{B} \odot \mathbf{C})) = ((\mathbf{B} \odot \mathbf{A}) \bullet \mathbf{C})$ and $(\mathbf{A} \bullet (\mathbf{BC})) = ((\mathbf{B}^T \mathbf{A}) \bullet \mathbf{C}) = ((\mathbf{A}\mathbf{C}^T) \bullet \mathbf{B})$ hold. Then put $\delta_Y = (\Delta_Y \bullet \mathbf{I}_{pq})$ and compute the partial derivative of J w.r.t. $\widetilde{\mathbf{C}}_{pq}$ as follows:

$$
\begin{aligned}
&\partial \mathrm{J}/\partial \widetilde{\mathbf{C}}_{pq} \\
&= (\mathbf{I}_2 \bullet (-(\widetilde{\mathbf{V}})_{\leq 1} \odot (\partial \widetilde{\mathbf{V}}/\partial \widetilde{\mathbf{C}}_{pq}))) + ((1-\mathbf{I}_2) \bullet ((\widetilde{\mathbf{V}})_{\geq 0} \odot (\partial \widetilde{\mathbf{V}}/\partial \widetilde{\mathbf{C}}_{pq}))) + \delta_Y \\
&= ((-(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2) \bullet (\partial \widetilde{\mathbf{V}}/\partial \widetilde{\mathbf{C}}_{pq})) + (((\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2)) \bullet (\partial \widetilde{\mathbf{V}}/\partial \widetilde{\mathbf{C}}_{pq})) + \delta_Y \\
&= ((-(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2 + (\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2)) \bullet (\widetilde{\mathbf{D}}(\partial \widetilde{\mathbf{M}}/\partial \widetilde{\mathbf{C}}_{pq}))) + \delta_Y \\
&= ((-(\widetilde{\mathbf{N}})_{\leq 1} \odot (\widetilde{\mathbf{D}}^T(-(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2 + (\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2))))(1-[\mathbf{I}_1 \; (1-\mathbf{I}_1)])^T \bullet \mathbf{I}_{pq}) \\
&+ (\Delta_Y \bullet \mathbf{I}_{pq}) \\
&= (((-(\widetilde{\mathbf{N}})_{\leq 1} \odot (\widetilde{\mathbf{D}}^T \widetilde{\mathbf{W}}))(1-[\mathbf{I}_1 \; (1-\mathbf{I}_1)])^T + \Delta_Y) \bullet \mathbf{I}_{pq})
\end{aligned}
$$

Since p, q are arbitrary, we have

$$
\begin{aligned}
\mathbf{J}_a^{\tilde{C}} =& \partial \mathrm{J}/\partial \widetilde{\mathbf{C}} \\
=& (-(\widetilde{\mathbf{N}})_{\leq 1} \odot (\widetilde{\mathbf{D}}^T \widetilde{\mathbf{W}}))(1-[\mathbf{I}_1 \; (1-\mathbf{I}_1)])^T + \Delta_Y \\
& \text{where } \widetilde{\mathbf{W}} = -(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2 + (\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2).
\end{aligned}
$$

Next we derive $\mathbf{J}_a^{\tilde{D}} = \partial \mathrm{J}/\partial \widetilde{\mathbf{D}}$ similarly. Put $\Delta_Z = (1-2\widetilde{\mathbf{D}}) \odot \mathbf{Z}$ and $\delta_Z = (\Delta_Z \bullet \mathbf{I}_{pq})$. Then for arbitrary p,q, we see

$$
\begin{aligned}
\partial \mathrm{J}/\partial \widetilde{\mathbf{D}}_{pq} =& (\mathbf{I}_2 \bullet -\partial \min_1(\widetilde{\mathbf{V}})/\partial \widetilde{\mathbf{D}}_{pq}) + (1-\mathbf{I}_2 \bullet \partial \max_0(\widetilde{\mathbf{V}})/\partial \widetilde{\mathbf{D}}_{pq}) + \delta_Z \\
=& ((-(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2) + (\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2) \bullet \partial \widetilde{\mathbf{V}}/\partial \widetilde{\mathbf{D}}_{pq}) + \delta_Z \\
=& (((-(\widetilde{\mathbf{V}})_{\leq 1} \odot \mathbf{I}_2) + (\widetilde{\mathbf{V}})_{\geq 0} \odot (1-\mathbf{I}_2))\widetilde{\mathbf{M}}^T \bullet \mathbf{I}_{pq}) + \delta_Z \\
=& (\widetilde{\mathbf{W}}\widetilde{\mathbf{M}}^T \bullet \mathbf{I}_{pq}) + (\Delta_Z \bullet \mathbf{I}_{pq}) \\
=& ((\widetilde{\mathbf{W}}\widetilde{\mathbf{M}}^T + \Delta_Z) \bullet \mathbf{I}_{pq}).
\end{aligned}
$$

So we reach $\mathbf{J}_a^{\tilde{D}} = \partial \mathrm{J}/\partial \widetilde{\mathbf{D}} = \widetilde{\mathbf{W}}\widetilde{\mathbf{M}}^T + \Delta_Z$. In actual learning, we use an adaptive gradient method Adam (Kingma and Ba, 2015) instead of gradient descent with a constant learning rate.

3.3 Learning algorithm

Given an input–output pair $(\mathbf{I}_1, \mathbf{I}_2)$ such that $f(\mathbf{I}_1) = \mathbf{I}_2$ for the target Boolean function f, Mat_DNF returns a matricized DNF $\varphi = (\mathbf{C}, \mathbf{D})$ giving $\varphi(\mathbf{I}_1) = \mathbf{I}_2$, basically by running **Algorithm 1** until $\mathrm{J} = 0$.

Algorithm 1 Mat_DNF

Input: binary matrix $\mathbf{I}_1(n \times l)$ of l interpretation vectors for n variables, row binary vector $\mathbf{I}_2(1 \times l)$ of Boolean output, learning rate α, integers h, max_try and max_itr

Output: matricized DNF $(\mathbf{C}(h \times 2n), \mathbf{D}(1 \times h))$, learning_error

1: initialize matrices $\widetilde{\mathbf{C}}(h \times 2n)$, $\widetilde{\mathbf{D}}(1 \times h)$ by Gaussian $\mathcal{N}(0, 1)$
2: **for all** $p = 1$ **to** max_try **do**
3: **for all** $q = 1$ **to** max_itr **do**
4: compute J by (3) and $\mathbf{J}_a^{\tilde{C}}$, $\mathbf{J}_a^{\tilde{D}}$ by (4)
5: update $\widetilde{\mathbf{C}}$, $\widetilde{\mathbf{D}}$ by Adam with initial learning rate α
6: threshold $\widetilde{\mathbf{C}}$, $\widetilde{\mathbf{D}}$ optimally to binary matrices $\mathbf{C}, \mathbf{D}$
7: by computing $\mathbf{I}_2^* = (\mathbf{D}(1 - \min_1(\mathbf{C}[(1 - \mathbf{I}_1);\ \mathbf{I}_1])_{\geq 1}$
8: and learning_error $= \|\mathbf{I}_2 - \mathbf{I}_2^*\|_1$
9: **if** learning_error $= 0$ **then**
10: exit p-loop
11: **end if**
12: **end for**
13: **if** learning_error $= 0$ **then**
14: exit q-loop
15: **end if**
16: perturbate $\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}$ by (5)
17: **end for**
18: **return** $(\mathbf{C}, \mathbf{D})$, learning_error

We however take a practical approach of thresholding $(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}})$ to binary $(\mathbf{C},\ \mathbf{D})$ even before J = 0 is reached assuming J is small and $\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}}$ are close to binary matrices. In more detail, the inner q-loop in **Algorithm 1** below iteratively updates $(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}})$ at most *max_itr* times while thresholding them optimally to binary $(\mathbf{C}, \mathbf{D})$ (line 6,7,8)[4] and computing learning_error using them. If $\varphi = (\mathbf{C}, \mathbf{D})$ achieves learning_error = 0, it exits from the q-loop and p-loop and returns φ. If learning_error > 0 happens even after *max_itr* iterations, it restarts the next q-loop with $(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}})$ perturbated by (5) where Δ_a and Δ_b are matrices of the same size as $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ respectively. They are comprised of elements sampled from the standard normal distribution $\mathcal{N}(0, 1)$. The perturbated $\widetilde{\mathbf{C}}$ and $\widetilde{\mathbf{D}}$ are used as initial parameters in the next loop (line 16). This perturbation is intended to escape from a local minimum.

$$\begin{aligned} \widetilde{\mathbf{C}_0} &= \sqrt{2/(h \cdot 2n)}\Delta_a + 0.5, \quad \widetilde{\mathbf{C}} = 0.5 \cdot (\widetilde{\mathbf{C}} + \widetilde{\mathbf{C}_0}) \\ \widetilde{\mathbf{D}_0} &= \sqrt{2/h}\Delta_b + 0.5, \quad \widetilde{\mathbf{D}} = 0.5 \cdot (\widetilde{\mathbf{D}} + \widetilde{\mathbf{D}_0}) \end{aligned} \tag{5}$$

[4] For example, $\widetilde{\mathbf{C}}$ is thresholded into $(\widetilde{\mathbf{C}})_{\geq\theta}$ where θ is between the maximum and minimum elements of $\widetilde{\mathbf{C}}$. We choose the best θ by trying 10 different θ's that gives the least learning_error.

Restart is allowed at most *max_try* times. Note that Mat_DNF possibly fails to achieve learning_error = 0 within given h, *max_itr* and *max_try*,[5] but when Mat_DNF returns a matricized DNF $\varphi = (\mathbf{C}, \mathbf{D})$ with learning_error = 0, it is guaranteed that J(**C**,**D**) = 0 and $\varphi(\mathbf{I}_1) = \mathbf{I}_2$ hold.

4 Learning as logical interpolation: a logical perspective

Here we characterize the learning of DNF φ by Mat_DNF from a logical perspective. Write $\models \phi_1 \Rightarrow \phi_2$ if $\phi_1 \Rightarrow \phi_2$ is a tautology. If we also have $\models \phi_2 \Rightarrow \phi_3$, ϕ_2 is called an interpolant between ϕ_1 and ϕ_3. Roughly, Craig's interpolation theorem (Craig, 1957) in first order logic states the existence of such interpolant. We prove that our learning of φ from an input–output pair $(\mathbf{I}_1, \mathbf{I}_2)$ such that $\varphi(\mathbf{I}_1) = \mathbf{I}_2$ is logically viewed as an inference of an interpolant φ.[6]

Suppose $(\mathbf{I}_1, \mathbf{I}_2)$ is an input–output pair for some n-variable Boolean function f and $f(\mathbf{I}_1) = \mathbf{I}_2$ holds. We divide the input binary matrix $\mathbf{I}_1(n \times l)$ into two submatrices $\mathbf{I}_1^P(n \times l_P)$ and $\mathbf{I}_1^N(n \times l_N)$ where $l_P + l_N = l$. $\mathbf{I}_1^P$ represents the positive (resp. negative) data and if $\mathbf{x} \in \mathbf{I}_1^P$ (resp. $\mathbf{x} \in \mathbf{I}_1^N$), $f(\mathbf{x}) = 1$ (resp. $f(\mathbf{x}) = 0$) holds.

We consider $\mathbf{I}_1^P$ as full DNF, DNF($\mathbf{I}_1^P$) in notation, in the following way. Let $\mathbf{x}$ be an interpretation vector in $\mathbf{I}_1$. Introduce conj($\mathbf{x}$) denoting a conjunction $l_1 \wedge \cdots \wedge l_n$ of literals such that $l_j = x_j$ if $\mathbf{x}(j) = 1$, else $l_j = \neg x_j$ $(1 \le j \le n)$. For example if $\mathbf{x} = [1\ 0\ 1]^T$, conj($\mathbf{x}$) $= x_1 \wedge \neg x_2 \wedge x_3$. Put DNF($\mathbf{I}_1^P$) $= \bigvee_{\mathbf{x} \in \mathbf{I}_1^P}$ conj($\mathbf{x}$) and call it the positive DNF for $(\mathbf{I}_1, \mathbf{I}_2)$. Likewise we define DNF($\mathbf{I}_1^N$) $= \bigvee_{\mathbf{x} \in \mathbf{I}_1^N}$ conj($\mathbf{x}$) and call it the negative DNF for $(\mathbf{I}_1, \mathbf{I}_2)$. For simplicity, we equate DNF($\mathbf{I}_1^P$) and DNF($\mathbf{I}_1^N$) respectively with the positive data $\mathbf{I}_1^P$ and negative data $\mathbf{I}_1^N$.

Proposition 2 *Let* $(\mathbf{I}_1, \mathbf{I}_2)$ *be an input–output pair for a Boolean function* f *such that* $f(\mathbf{I}_1) = \mathbf{I}_2$. *Also let* DNF($\mathbf{I}_1^P$) *and* DNF($\mathbf{I}_1^N$) *respectively be the positive and negative DNF for* $(\mathbf{I}_1, \mathbf{I}_2)$. *For a DNF* φ, $\varphi(\mathbf{I}_1) = \mathbf{I}_2$ *if-and-only-if* φ *is an interpolant between* DNF($\mathbf{I}_1^P$) *and* $\neg$ DNF ($\mathbf{I}_1^N$).

Proof We first prove the only-if part. Suppose $\varphi(\mathbf{I}_1) = \mathbf{I}_2$. Let $\mathbf{i}$ be an interpretation vector over n variables satisfying DNF($\mathbf{I}_1^P$). It satisfies some disjunct conj($\mathbf{x}$) in DNF($\mathbf{I}_1^P$). Since conj($\mathbf{x}$) is a conjunction of n distinct literals, the fact that $\mathbf{i}$ satisfies conj($\mathbf{x}$) implies $\mathbf{i} = \mathbf{x}$ as vector. On the other hand, we have $\varphi(\mathbf{I}_1) = \mathbf{I}_2 = f(\mathbf{I}_1)$ by assumption and hence $\varphi(\mathbf{x}) = f(\mathbf{x})$ as $\mathbf{x} \in \mathbf{I}_1^P \subseteq \mathbf{I}_1$. We also have $f(\mathbf{x}) = 1$ as $\mathbf{x} \in \mathbf{I}_1^P$. Putting the two together, we conclude $\varphi(\mathbf{i}) = \varphi(\mathbf{x}) = f(\mathbf{x}) = 1$. Since $\mathbf{i}$ is arbitrary and satisfies φ, $\models$ DNF($\mathbf{I}_1^P$) $\Rightarrow \varphi$ is proved. $\models \varphi \Rightarrow \neg$DNF($\mathbf{I}_1^N$) is proved similarly by proving $\models$ DNF($\mathbf{I}_1^N$) $\Rightarrow \neg\varphi$.

To prove the if-part, recall that an interpolant φ satisfies $\models$ DNF($\mathbf{I}_1^P$) $\Rightarrow \varphi$ and $\models$ DNF($\mathbf{I}_1^N$) $\Rightarrow \neg\varphi$. So if $\mathbf{x} \in \mathbf{I}_1^P$ (resp. $\mathbf{x} \in \mathbf{I}_1^N$), then DNF($\mathbf{I}_1^P$)($\mathbf{x}$) = 1 and hence $\varphi(\mathbf{x}) = 1$ holds (resp. then DNF($\mathbf{I}_1^N$)($\mathbf{x}$) = 1 and hence $\varphi(\mathbf{x}) = 0$ holds). In other words, if $\mathbf{x} \in \mathbf{I}_1^P$, $\varphi(\mathbf{x}) = 1 = f(\mathbf{x})$ and if $\mathbf{x} \in \mathbf{I}_1^N$, $\varphi(\mathbf{x}) = 0 = f(\mathbf{x})$. So we reach $\varphi(\mathbf{I}_1) = f(\mathbf{I}_1) = \mathbf{I}_2$. □

[5] This happens, for example, when learning data is inconsistent and there is no Boolean function satisfying the learning data. It also can happen when the target function is difficult to learn as the case of the n-parity function with large n.

[6] Craig's interpolation theorem is the one for first-order logic but its propositional version has long been practically applied to model checking (Vizel et al., 2015; McMillan et al., 2018)

By **Proposition** 2, we can say that φ returned by Mat_DNF with learning_error = 0 is an interpolant between DNF($\mathbf{I}_1^P$) and ¬DNF($\mathbf{I}_1^N$). We can also say by combining **Proposition** 1 and 2 that finding a root of J($\mathbf{C}, \mathbf{D}$) = 0 defined by (3), learning a DNF φ satisfying $\varphi(\mathbf{I}_1) = \mathbf{I}_2$ and inferring an interpolant φ between DNF($\mathbf{I}_1^P$) and ¬DNF($\mathbf{I}_1^N$) are one and the same thing, they are all equivalent.

The recognition of this equivalence has some interesting consequences. The first one is that from the viewpoint of classification, learning by Mat_DNF consists of learning the feature space of conjunctions $\widetilde{\mathbf{C}}$ and its linear separation by a hyperplane specified by a continuous disjunction $\widetilde{\mathbf{D}}$ as shown in the equation (2). Hence it seems possible to modify Mat_DNF so that it can search for a "max-margin interpolant" corresponding to the max-merging hyperplane, which is expected to generalize well. Sharma et. al already proposed to use SVM to infer interpolants (Sharma et al., 2012) where SVM is applied to the predefined feature space. In our "max-margin interpolant" inference, if realized, the feature space itself will be learned by Mat_DNF.

The second one is the possibility of a neural end-to-end refutation prover. Let S be a set of ground clauses. Also let $S = S_1 \cup S_2$ be any split of S such that atom(S_1) ∩ atom(S_2) ≠ ∅ where atom(S_i) denotes the set of atoms in S_i ($i = 1, 2$). It can be proved that S is unsatisfiable if-and-only-if there is an interpolant φ between S_1 and ¬S_2 (proof omitted as it is out of the scope of this paper (Vizel et al., 2015; McMillan et al., 2018)). We can apply Mat_DNF to infer this φ assuming that S_1 is positive data (φ is true over S_1) and S_2 is negative data (φ is false over S_2) respectively.

The third one concerns the generalizability of the DNF φ learned by Mat_DNF. It is observed that φ tends to overgeneralize positive data $\mathbf{I}_1^P$ in the input data. That is, $\models$ DNF($\mathbf{I}_1^P$) → φ holds but sometimes the degree of generalization by logical implication measured by the distance between DNF($\mathbf{I}_1^P$) and φ is too high, which adversely affects the accuracy of φ. Later in Sect. 5.5, we propose a way of controlling the distance between DNF($\mathbf{I}_1^P$) and φ and show that the accuracy of φ is actually improved.

5 Learning random DNFs

5.1 Performance measures and generalization

First we define some performance measures concerning Mat_DNF to clarify the meaning of generalization. Let f be a target Boolean function in n variables, $\mathbf{I}_0$ the domain matrix for n variables and ($\mathbf{I}_1$,$f(\mathbf{I}_1)$) ($\mathbf{I}_1 \subseteq \mathbf{I}_0$) an input–output pair for f supplied as learning data for Mat_DNF. We introduce "domain ratio" $dr = \frac{|\mathbf{I}_1|}{|\mathbf{I}_0|}$ ($0 \leq dr \leq 1$) where $|\,\mathbf{I}\,|$ denotes the number of interpretation vectors in $\mathbf{I}$. Domain ratio dr is the relative size of learning data to the whole domain data. In what follows, purely for convenience, we use dr even when $dr \cdot |\,\mathbf{I}_0\,|$ is not an integer. In such case, it means $\mathbf{I}_1$ contains the $\lfloor dr \cdot |\,\mathbf{I}_0\,| \rfloor$ number of interpretation vectors of $\mathbf{I}_0$.

Suppose we have obtained a DNF $\varphi = (\mathbf{C}, \mathbf{D})$ with learning_error = 0 by running Mat_DNF on ($\mathbf{I}_1$,$f(\mathbf{I}_1)$). Compute $\varphi(\mathbf{I}_0) = (\mathbf{D}(1 - \min_1(\mathbf{C}[(1 - \mathbf{I}_0); \mathbf{I}_0])_{\geq 1}$ (see (1)) and exact_error = $\|f(\mathbf{I}_0) - \varphi(\mathbf{I}_0)\|_1$ which is the number of different bits between $f(\mathbf{I}_0)$ and $\varphi(\mathbf{I}_0)$. Introduce acc_DNF, the "exact accuracy" of φ, by defining acc_DNF = $1 - \text{exact\_error}/2^n$. Since learning_error is zero, φ perfectly reproduces $f(\mathbf{I}_1)$ and hence it follows that acc_DNF = $dr + (1 - dr) \cdot$ acc_pred where acc_pred is the prediction accuracy of φ over the unseen domain data $\mathbf{I}_0 \backslash \mathbf{I}_1$ not used for learning.

Consequently we have acc_pred = (acc_DNF − dr)/(1 − dr). Thus prediction accuracy and exact accuracy are mutually convertible. Finally we define generalization. Introduce acc_dr = $dr + 0.5 \cdot (1 - dr) = 0.5 \cdot (1 + dr)$ which is the expected accuracy of a base line learner learning data with domain ratio dr that completely memorizes learning data (dr) and makes a random guess on unseen data ($0.5 \cdot (1 - dr)$). We say *generalization occurs* when acc_DNF > acc_dr = $0.5 \cdot (1 + dr)$, or equivalently acc_pred > 0.5 holds (because acc_DNF − acc_dr = $(1 - dr) \cdot$ (acc_pred − 0.5)).

5.2 Measuring accuracy for random DNFs

We conduct a learning experiment with small random DNFs to examine the learning behavior of Mat_DNF w.r.t. data scarcity controlled by domain ratio dr and see how generalization occurs[7,8] We first randomly generate a DNF φ_0 in $n = 5$ variables that consists of three disjuncts, each containing at most 5 lals a half of which is negative on average. We also generate a domain matrix $\mathbf{I}_0(n \times 2^n)$ for $n = 5$ variables. Next suppose a domain ratio dr is given. For this dr, we generate a binary matrix $\mathbf{I}_1(n \times l)$ consisting of $l = 2^n \cdot dr$ interpretation vectors randomly sampled without replacement from $\mathbf{I}_0$. Then we run Mat_DNF on the learning data $(\mathbf{I}_1, \varphi_0(\mathbf{I}_1))$[9] and obtain a DNF φ_1 that perfectly classifies the learning data, i.e. $\varphi_1(\mathbf{I}_1) = \varphi_0(\mathbf{I}_1)$ and compute the exact accuracy acc_DNF of φ_1. We repeat this process 100 times and obtain the average acc_DNF of φ_1 against dr.

By varying $dr \in \{0.1, \ldots, 1.0\}$, we obtain a curve of exact accuracy w.r.t. dr denoted as acc_DNF in Fig. 1. There acc_dr denotes the expected accuracy of the base line learner performing only memorization and random guess. Other two curves, acc_DNF_noise and acc_over, are explained next. We observe that acc_DNF is always (and slightly) above acc_dr for all dr's. So this experiment confirms that generalization in our sense actually occurs and the learned DNF does more than just pure memorization and random guess by detecting some logical pattern.

5.3 Noise-expansion and over-iteration

The acc_DNF_noise and acc_DNF_over curves in Fig. 1 demonstrate that generalization occurs with a greater degree than acc_DNF, i.e. acc_DNF_noise ≈ acc_DNF_over > acc_DNF holds at most dr's. They are obtained by two different operations, acc_DNF_noise by "noise-expansion" and acc_DNF_over by "over-iteration", respectively.

The first operation, noise-expansion, means the expansion of an input vector in the learning data $\mathbf{I}_1$ by a random bit vector. For example, a 5 bit input vector $\mathbf{x} = [0\,1\,0\,1\,0]^T$ in $\mathbf{I}_1$ is expanded into a 10 dimensional vector $\mathbf{x}_{\text{noise}} = [\mathbf{x}\mathbf{n}] = [0\,1\,0\,1\,0\,1\,0\,0\,1\,1]^T$ by appending a random bit vector $\mathbf{n} = [1\,0\,0\,1\,1]^T$ to $\mathbf{x}$. In learning, each $\mathbf{x}$ in $\mathbf{I}_1$ is expanded

[7] All programs used in this paper are written in GNU Octave 4.2.2 and run on a PC with Intel(R) Core(TM) i7-10700@2.90GHz CPU with 26GB memory. Due to the naive nature of our implementation of Mat_DNF, the experiment scale is small.

[8] We also implemented Mat_DNF by PyTorch and conducted a learning experiment for the 7-parity function from complete data. We chose the parity function because it is known to be hard to learn. As average over 5 trials, the PyTorch version took 42.9 s(10.5) on Google Colaboratory (GPU) while the octave version (CPU) took only 9.6 s(11.5). Although the difference may be due to our naive use of PyTorch, it seems likely that our matrix-based implementation is suitable for Octave.

[9] The learning parameters are set to $\alpha = 0.1$, $max\_try = 20$, $max\_itr = 500$ and $h = 1000$.

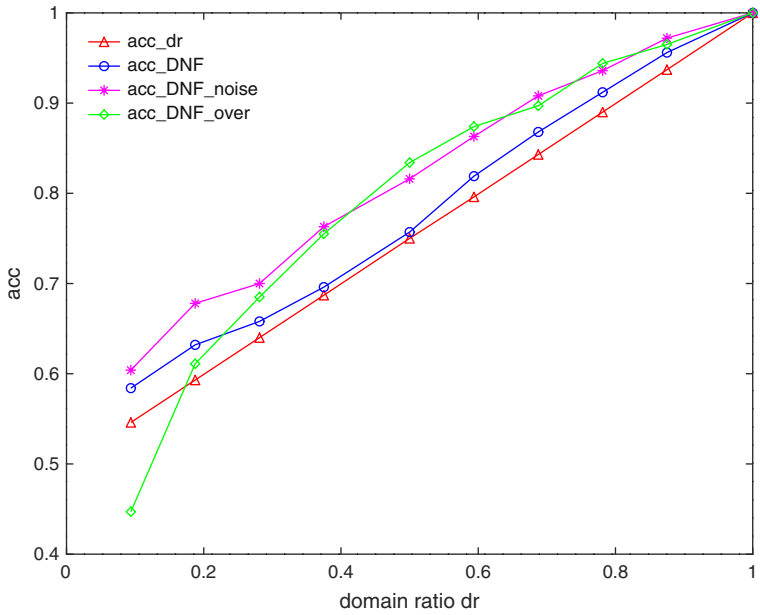

Fig. 1 "exact accuracy" of DNF learned with noise-expansion and over-iteration

into $\mathbf{x}_{\text{noise}}$ and then used for learning. Although each input vector in $\mathbf{I}_1$ gets longer (length doubled) by noise-expansion, the number of input vectors remains the same. It simply means Mat_DNF has an additional task of identifying those variables in an input vector $\mathbf{x}_{\text{noise}}$ that are relevant to the output, hereby causing additional update steps in **Algorithm 1**. So from the viewpoint of minimizing J to zero, the net effect of noise-expansion is to force Mat_DNF to find another root of J even when $\text{J} = 0$ is reached in the original learning task. This point is made clear by comparing with "over-iteration" explained below.

The second operation, over-iteration, forces Mat_DNF to skip a root of $\text{J} = 0$ found first and keep learning. Only after some prespecified extra steps (for example extra_update = 20 in the case of acc_DNF_itr in Fig. 1) have been made, Mat_DNF is allowed to return when a root of J is found again. Intuitively, this operation have the effect of avoiding a root near the initializing point that often overfits the learning data and exploring a root in the relatively flat landscape of J. In other words, over-iteration searcher for a root of J closer to a global minimum such as the target DNF.

Observe that as the acc_DNF_noise and acc_DNF_over curves in Fig. 1 show, not only both noise-expansion and over-iteration improve exact accuracy, or equivalently prediction accuracy, but with a similar degree of improvement. Hence it seems reasonable to hypothesize that noise-expansion causes over-iteration and over-iteration causes the improvement of exact accuracy.

The result of this experiment also indicates the importance of an intentional choice of a local minimum (choosing a root in our case) which is independently suggested by "flooding" (Ishida et al., 2020) and "grokking" (Power et al., 2021). In flooding, learning is controlled by gradient descent and ascent to keep training error small but non-zero. In grokking, learning is continued even after learning accuracy is saturated, and then test accuracy

Table 1 Domain ratio *dr*, distance and the probability of logical consequence and equivalence

dr	0.1	0.3	0.5	0.7	0.9	1.0
distance	13.6	11.8	7.5	4.2	1.4	0.0
distance_itr	10.9	6.7	4.2	2.1	0.7	0.0
p_conseq	0.54	0.88	0.94	0.89	0.92	1.0
p_conseq_itr	0.24	0.16	0.30	0.58	0.84	1.0
p_equiv	0.01	0.00	0.03	0.02	0.20	1.0
p_equiv_itr	0.00	0.01	0.06	**0.19**	**0.55**	1.0

suddenly rises to a high level. Our over-iteration has a similar effect of moving around local minima in a flat loss landscape, leading to better generalization.

5.4 The logical relations and over-iteration

When a learning target is a DNF φ_0, we naturally ask a logical question of whether the consequence relation and equivalence relation between φ_0 and a learned DNF φ hold or not. We also interested in the distance between them[10] because we expect φ to be logically related to φ_0 when φ is close to φ_0. So we estimate the probability p_conseq (resp. p_equiv) of φ being a logical consequence of φ_0, i.e., $\models \varphi_0 \Rightarrow \varphi$ in notation (resp. φ being logically equivalent to φ_0, i.e., $\models \varphi_0 \Leftrightarrow \varphi$) for a 5-variable DNF φ_0 generated as in the previous section, together with the average distance between φ_0 and φ by running Mat_DNF 100 times[11] and counting the number of runs that make these logical relations hold and computing the average distance. We obtain Table 1.

In Table 1, distance_itr is the same as distance between the target DNF φ_0 and a learned DNF φ but obtained by over-iteration with extra_update = 60. The same applies for p_equiv_itr and p_equiv.

First we can recognize in the table that larger data gives us a more exact solution. That is, the distance between the target DNF φ_0 and a learned DNF φ monotonically decreases as *dr* gets closer to 1. Furthermore the effect of over-iteration is clearly visible. It gets the learned DNF much closer to the target DNF, from 7.5 to 4.2 at $dr = 0.5$ for example. In other words, it chooses a root of the cost function J near the target φ_0.

Concerning logical relations, observe that p_conseq and p_equiv in Table 1 more or less monotonically increase as *dr* increases. So again, larger data gives a bigger chance of the logical relationship. Second observe that p_conseq, the probability of $\models \varphi_0 \Rightarrow \varphi$, is rather high through all *dr*'s but lowered considerably by over-iteration. Third over-iteration has the opposite effect on p_equiv, the probability of $\models \varphi_0 \Leftrightarrow \varphi$ however. It greatly improves the chance of $\models \varphi_0 \Leftrightarrow \varphi$ after $dr > 0.5$. For example, p_equiv suddenly jumps up from 0.02 to 0.19 at $dr = 0.7$ and from 0.20 to 0.55 at $dr = 0.9$ (see bold figures in Table 1). This positive effect of over-iteration on p_equiv becomes critical when applying Mat_DNF to Boolean network learning. This is because the primary purpose of our Boolean network

[10] The distance between φ_0 and φ in n variables is defined to be the number of interpretation vectors $\mathbf{x}$ in the domain matrix for n variables such that $\varphi_0(\mathbf{x}) \neq \varphi(\mathbf{x})$.

[11] Learning parameters are $\alpha = 0.1$, $max\_try = 20$, $max\_itr = 500$ and $h = 1000$.

learning is to recover the original DNFs in the target Boolean network and over-iteration in this section enhances the chance of discovering such DNFs.

5.5 Controlling logical generalization

Over-iteration wanders in the search space for a better local minimum. Here we introduce another more proactive approach for the same purpose based on **Proposition** 2 in Sect. 4. This approach has the sense of search direction, away from negative data and toward positive data, thus making it possible to control the degree of generalization of the learned DNF.

Let φ_0 be a target DNF, $\mathbf{I}_0$ the domain of φ_0, $(\mathbf{I}_1, \mathbf{I}_2)$ an input–output pair for learning where $\mathbf{I}_1 \subseteq \mathbf{I}_0$ and $\mathbf{I}_2 = \varphi_0(I_1)$. Also let DNF($\mathbf{I}_1^P$) and DNF($\mathbf{I}_1^N$) respectively be the positive and negative DNF for $(\mathbf{I}_1, \mathbf{I}_2)$ introduced in Sect. 4 associated with the positive data $\mathbf{I}_1^P$ and negative data $\mathbf{I}_1^N$ in $\mathbf{I}_1$.

Our idea is based on the empirical observation that when learning random DNFs form insufficient data by Mat_DNF, despite the fact that the target DNF φ_0 and the learned DNF φ are both interpolants between the DNF($\mathbf{I}_1^P$) and ¬DNF($\mathbf{I}_1^N$) according to **Proposition** 2, their distance to DNF($\mathbf{I}_1^P$) and DNF($\mathbf{I}_1^N$) often differs greatly. Since learning data is randomly generated using the target DNF φ_0, usually φ_0 is located (almost) in the middle between DNF($\mathbf{I}_1^P$) and DNF($\mathbf{I}_1^N$) distance-wise. However, it is observed that the learned φ is very close to the negative data DNF($\mathbf{I}_1^N$). In other words, due to the learning bias of Mat_DNF, φ tends to overgeneralize positive data by yielding disjuncts outside the original positive data DNF($\mathbf{I}_1^P$).

To combat this overgeneralization of positive data by Mat_DNF, we add a special term J_{int} to the cost function J to suppress the generation of disjuncts in φ. Concretely J_{int} is computed as follows.

$$
\begin{aligned}
\mathbf{I}_0^P &= \mathbf{I}_0 \setminus \mathbf{I}_1^P \\
\widetilde{\mathbf{N}}^P &= \widetilde{\mathbf{C}}[(1 - \mathbf{I}_0^P); \mathbf{I}_0^P] \\
\widetilde{\mathbf{M}}^P &= 1 - \min_1(\widetilde{\mathbf{N}}^P) \\
\mathrm{J}_{int} &= \sum \max_0(\widetilde{\mathbf{D}}\widetilde{\mathbf{M}}^P)
\end{aligned}
$$

Here $\mathbf{I}_0^P$ is the set of interpretation vectors which, when considered as conjunctions, can be added to DNF($\mathbf{I}_1^P$) as disjuncts in the learned φ. $\widetilde{\mathbf{M}}^P$ is the truth values of continuous conjunctions represented by $\widetilde{\mathbf{C}}$. $\widetilde{\mathbf{D}}\widetilde{\mathbf{M}}^P$ is the truth values of the continuous DNF $(\widetilde{\mathbf{C}}, \widetilde{\mathbf{D}})$ evaluated by the interpretation vectors $\mathbf{I}_0^P$. Minimizing J_{int} causes minimizing positive elements in $\widetilde{\mathbf{D}}\widetilde{\mathbf{M}}^P$ sifted out by $\max_0(\cdot)$ to zero, in which case, as $\widetilde{\mathbf{M}}^P$ is non-negative, pushing positive elements in $\widetilde{\mathbf{D}}$ to zero, leading to a small number of disjuncts in the thresholded disjunction D in φ, i.e. a small number of disjuncts in φ.

We conduct a learning experiment of the 5-ary random DNF with this penalty term J_{int} added to the cost function J in the form of $\beta \cdot \mathrm{J}_{int}$ $(\beta \geq 0)$ while varying β from 0 to 5.[12] We choose $dr = 0.5$ and randomly generate a target DNF φ_0 and the learning data $(\mathbf{I}_1, \varphi_0(\mathbf{I}_1))$

[12] The experimental parameters for one trial are $\alpha = 0.1$, max_try = 20, max_itr = 500, h = 1000. No over-iteration is used.

Table 2 The effect of J_{int} on the learned φ

β	0.0	0.01	0.05	0.1	0.4	1.0	5.0
dist(DNF($\mathbf{I}_1^P$), φ)	14.3	12.9	9.6	8.7	3.5	2.5	2.2
dist(φ_0, φ)	7.6	6.5	**5.6**	5.9	6.3	6.3	5.9
exact accuracy	0.736	0.777	**0.824**	0.807	0.794	0.802	0.815

Bold numbers indicate the best results in this experiment

as in Sect. 5.2. So half of the complete data necessary for identifying the target φ_0 is supplied to the learner.

We run Mat_DNF on the learning data until learning error becomes zero and measure the exact accuracy of the learned DNF φ in each learning trial. Table 2 contains figures averaged over 100 trials[13].

Clearly as β gets larger (while $\models \text{DNF}(\mathbf{I}_1^P) \rightarrow \varphi$ is the same), the distance between the positive learning data DNF($\mathbf{I}_1^P$) and the learned DNF φ monotonically decreases, which verifies the effectiveness of the penalty term J_{int} to manipulate the degree of logical implication.

On the other hand, the distance between the target φ_0 and the learned φ draws a convex curve w.r.t. β and φ achieves the maximum exact accuracy 0.824 when dist(φ_0,φ) is the least 5.6. In other words, we can change the distance between the target DNF and learned DNF by a parameter β in vector spaces for better generalization.

6 Learning Boolean networks

We apply Mat_DNF to learning Boolean networks (BNs) introduced by Kauffman (Kauffman, 1969) which have been used to model gene regulatory networks in biology. A BN is biological network where nodes are genes with $\{0, 1\}$ states and a state transition (activation of gene expression) of a gene occurs according to a Boolean formula associated with it. The learning task is to infer Boolean formulas associated with nodes from state transition data. Due to the general hardness results of learning Boolean formulas (Feldman, 2007), BN learning on a large scale is difficult. We select three BNs of moderate size from literature for learning, one for mammalian cell cycle from Fauré et al. (2006), one for budding yeast cell cycle from Irons (2009) and one for myeloid differentiation from Krumsiek et al. (2011). Learning performance is evaluated in terms of the recovery rate of the original DNFs associated with a BN.

6.1 Learning a mammalian cell cycle BN

In the first learning experiment, we use a synchronous BN for mammalian cell cycle having 10 nodes (genes) (Fauré et al., 2006) where state transition occurs simultaneously for all genes. A state of the BN is represented by a state vector $\mathbf{x} \in \{0, 1\}^{10}$ and a state of each gene_i is described by a Boolean variable x_i $(1 \leq i \leq 10)$ and its

[13] Recall that for Boolean formulas A, B in n variables, the distance between A and B is given by dist(A, B) $= | \{\mathbf{x} \in \mathbf{I}_0 \mid \mathbf{x} \models (A \wedge \neg B) \vee (\neg A \wedge B)\} |$ where $\mathbf{I}_0$ is the set of 2^n interpretation vectors.

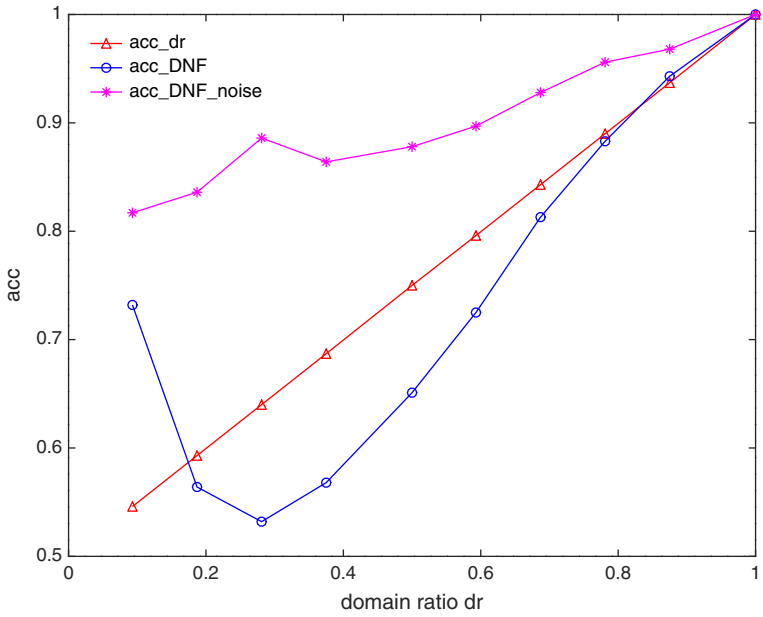

Fig. 2 "exact accuracy" of DNF learned from ϕ_6 with noise-expansion

state by $\mathbf{x}(i) \in \{1,0\}$. A state transition of gene_i is controlled by a DNF ϕ_i associated with it, i.e. the next state of gene_i= 1 if $\phi_i(\mathbf{x}) = 1$, otherwise gene_i= 0. We obtain from Fauré et al. (2006) 10 DNFs associated with 10 genes. For example $\phi_6 = (\neg x_1 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_4 \wedge x_6 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10})$ is associated with gene_6.

To see to what degree Mat_DNF can recover the original 10 DNFs, following (Inoue et al., 2014), we consider ϕ_i $(1 \leq i \leq 10)$ as a 10-variable Boolean function and prepare as learning data a complete input–output pair $(\mathbf{I}_0^{10}, \phi_i(\mathbf{I}_0^{(10)}))$ for ϕ_i where $\mathbf{I}_0^{(10)}$ is the domain matrix for 10 variables containing 1024 interpretation vectors. Then we let Mat_DNF learn a DNF φ from $(\mathbf{I}_0^{(10)}, \phi_i(\mathbf{I}_0^{(10)}))$[14] and check if φ is identical to the original ϕ_i. The result is encouraging. Nine DNFs out of the original 10 DNFs are successfully recovered (modulo renaming) and the remaining one is logically equivalent to the original DNF.

To understand the origin of this high recovery rate, we pick up a DNF ϕ_6 associated with gene_6 and examine noise-expansion effect on it. We consider ϕ_6 as a 5-variable Boolean function over the domain matrix $\mathbf{I}_0^{(5)}$ and measure acc_DNF w.r.t. dr. To measure acc_DNF_noise, we append a 5 dimensional random bit vector to each interpretation vector in $\mathbf{I}_0^{(5)}$. The learning result is shown in Fig. 2 where figures are the average over 100 trials. There we see the acc_DNF curve shows a large improvement in acc_DNF by noise-expansion compared to the case of Fig. 1. For example it achieves acc_DNF = 0.817 at $dr = 0.1$, which means on average, given only 3 input–output pairs, Mat_DNF learns by noise-expansion a DNF that correctly predicts 26 input–output pairs in $(\mathbf{I}_0^{(5)}, \phi_6(\mathbf{I}_0^{(5)}))$ out of

[14] Learning parameters are set to $\alpha = 0.1$, $max\_itr = 500$, $max\_try = 50$ and $h = 1000$.

Table 3 Examples of learned DNFs learned from ϕ_6

dr	Learned DNF	Relation to ϕ_6
1.0	$(\neg x_1 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_4 \wedge x_6 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10})$	Identical
0.8	$(\neg x_1 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge \neg x_4 \wedge x_5 \wedge x_6 \wedge \neg x_{10})$	Equivalent
0.5	$(\neg x_1 \wedge \neg x_4 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge \neg x_4 \wedge x_6 \wedge \neg x_{10})$	Equivalent
0.3	$(\neg x_1 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_{10}) \vee (\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge \neg x_4 \wedge x_5 \wedge x_6 \wedge \neg x_{10}) \vee (\neg x_1 \wedge x_3 \wedge \neg x_4 \wedge x_8 \wedge x_9 \wedge \neg x_{10})$	Consequence
0.1	$(\neg x_1 \wedge \neg x_2 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge x_3 \wedge \neg x_4 \wedge \neg x_8 \wedge \neg x_9 \wedge \neg x_{10})$ $\vee(\neg x_1 \wedge \neg x_5 \wedge x_6 \wedge \neg x_{10}) \vee (\neg x_1 \wedge x_2 \wedge \neg x_5 \wedge \neg x_{10})$	Independent

32 possible tests. Such high accuracies plotted in Fig. 2 strongly suggests that noise-expansion helps Mat_DNF find a DNF with high generalizability, or the original DNF. Also we can point out that the big difference in the effect of noise-expansion between Fig. 1 and Fig. 2 might be attributed to the nature of the learning target ϕ_6 which is not randomly generated but comes from biological literature.

Then look at the learning experiment of mammalian cell cycle BN again. Note that although ϕ_6 is a function of 5 variables $\{x_1, x_4, x_5, x_6, x_{10}\}$, it is treated as a function of 10 variables $\{x_1, \ldots, x_{10}\}$ in the experiment. So the remaining 5 variables $\{x_2, x_3, x_7, x_8, x_9\}$ behave as noise bits in learning just like noise-expansion. This implicit noise-expansion happens to the learning of all DNFs $\{\phi_1, \ldots, \phi_{10}\}$ because they contain only at most 6 variables. Moreover, since they are not random DNFs, noise-expansion can be particularly effective as shown in Fig. 2, and hence it is not unreasonable to assume that Mat_DNF is likely to able to learn the original DNFs, which explains the high recovery rate of the original DNFs.

We conclude this section by looking at DNFs learned from insufficient data to develop an insight into the syntactic aspect of learned DNFs and their logical relationship to the target DNF. Table 3 lists some DNFs learned from an input-out pair for ϕ_6 obtained by applying ϕ_6 as a 10-variable function to the interpretation vectors of size $2^{10} \times dr$ sampled without replacement from the domain matrix $\mathbf{I}_0^{(10)}$.[15]

In Table 3, for $dr \in \{1.0, 0.8, 0.5\}$, every data used for learning contains 32 different input–output pairs, i.e. contains complete information about ϕ_6. That is why all learned DNFs are logically equivalent to ϕ_6. At $dr = 0.3$, learning data still contains all information on ϕ_6. Nonetheless the learned DNF have extraneous variables not appearing in the original $\phi_6(x_1, x_4, x_5, x_6, x_{10})$ which destroy the logical equivalence to ϕ_6 though it still continues to be a logical consequence. When dr is further lowered to $dr = 0.1$, constraint by learning data is more loosened. So more conjunctions and extraneous variables are introduced to the learned DNF and they stop the learned DNF from being either a logical consequence of or logically equivalent to ϕ_6.

[15] Learning parameters are $\alpha = 0.1$, $max\_try = 20$, $max\_itr = 500$ and $h = 1000$.

Table 4 Recovered Boolean formulas for the asynchronous myeloid differentiation BN

Target gene	Fact	rfBFE	Mat_DNF
GATA-2	GATA-2 ∧ ¬(GATA-1 ∧ FOG-1) ∧ ¬ PU.1	¬FOG-1 ∧ ¬PU.1	¬FOG-1 ∧ ¬PU.1
GATA-1	(GATA-1 ∨ GATA-2 ∨ Fli-1) ∧ ¬PU.1	¬PU.1	¬PU.1
FOG-1	GATA-1	GATA-1	GATA-1
EKLF	GATA-1 ∧ ¬Fli-1	GATA-1 ∧ ¬Fli-1	GATA-1 ∧ ¬Fli-1
Fli-1	GATA-1 ∧ ¬EKLF	GATA-1 ∧ ¬EKLF	GATA-1 ∧ ¬EKLF
SCL	GATA-1 ∧ ¬PU.1	GATA-1	GATA-1
C/EBPa	C/EBPa ∧ ¬(GATA-1 ∧ FOG-1 ∧ SCL)	¬FOG-1 ∨ ¬SCL	¬FOG-1 ∨ ¬SCL
PU.1	(C/EBPa ∨ PU.1) ∧ ¬(GATA-1 ∨ GATA-2)	¬GATA-2 ∧ PU.1	¬GATA-2 ∧ ¬GATA-1 ∧ PU.1
cJun	PU.1 ∧ ¬Gfi-1	PU.1 ∧ ¬Gfi-1	PU.1 ∧ ¬Gfi-1
EgrNab	(PU.1 ∧ cJun) ∧ ¬Gfi-1	PU.1 ∧ cJun ∧ ¬Gfi-1	PU.1 ∧ cJun ∧ ¬Gfi-1
Gfi-1	C/EBPa ∧ ¬EgrNab	C/EBPa ∧ ¬EgrNab	C/EBPa ∧ ¬EgrNab

6.2 Learning a budding yeast cell cycle BN

We conduct the second experiment with a synchronous BN for budding yeast cell cycle taken from Irons (2009). Since it contains 18 genes (DNFs) and preparing gene expression data is very time-consuming, it is unrealistic to assume the whole domain matrix $\mathbf{I}_0^{(18)}$ containing $2^{18} = 262,144$ data points as learning data to learn a Boolean formula ϕ_i for gene_i in the BN (Irons, 2009) ($1 \leq i \leq 18$).

We instead randomly generate a set of state vectors $\mathbf{I}_1^{\text{rand}}$ of size 1, 000 and use $(\mathbf{I}_1^{\text{rand}}, \phi_i(\mathbf{I}_1^{\text{rand}}))$ ($1 \leq i \leq 18$) as learning data to learn a DNF for ϕ_i.[16]

In this experiment, 17 DNFs out the 18 original DNFs are successfully recovered in at most three trials and the remaining DNF is logically equivalent to the original one. Considering the severe data scarcity such that only 0.38% ($1000/2^{18}$) of the whole data is supplied as learning data, this success rate is somewhat surprising, but again can be explained as the effect of implicit noise-expansion as in the mammalian cell cycle case because the set of variables relevant to a target gene is surely a proper subset of 18 variables and the remaining irrelevant ones would behave as noise.

6.3 Learning a myeloid differentiation BN

The last example is learning an asynchronous BN with 11 genes for myeloid differentiation process (Krumsiek et al., 2011). In this "biologically more feasible" BN (Gao et al., 2018), state transition occurs asynchronously where a gene is nondeterministically chosen and the Boolean function (DNF) associated with the gene is applied to the current state to decide the next state of the BN.

[16] Learning parameters are $\alpha = 0.1$, $max\_try = 20$, $max\_itr = 500$ and $h = 2000$.

Following (Gao et al., 2018), we generate learning data for asynchronous BN by simulating all possible asynchronous sate transitions starting from an "early, unstable undifferentiated state, where only GATA-2, C/EBPa, and PU.1 are active" (Krumsiek et al., 2011). This simulation generates 160 distinct hierarchically layered states containing four point attractors that correspond to four mature blood cells. For each gene, we generate state transition data of size 160 from these states and let Mat_DNF learn it with over-iteration (extra_update = 100). Since a learned DNF varies with initialization, we repeat this asynchronous BN data learning ten times and consider the majority of ten learned DNFs as the learned DNF for the target gene.

Out of 11 DNFs to be recovered, Mat_DNF correctly recovered the original DNFs for 6 genes (Table 4). They are all pure conjunctions. DNFs for the remaining 5 genes are recovered partially in such a way that they lost at most three variables from the original ones. We performed other measurements.

We now compare our results with those by rfBFE (Gao et al., 2018) in more detail. rfBFE is one of the state-of-the-art BN learning algorithms which is a refinement of BestFit extension algorithm (Lähdesmäki et al., 2003)[17]. Since the purpose of BN learning is to infer Boolean formulas governing the state transitions process, the recovery rate of target Boolean formulas is the most important criterion. From this viewpoint, it is to be noted that when applied to complete data generated by synchronous BN, both rfBFE and Mat_DNF recover all original 11 DNFs. However there is a big difference in execution time. While rfBFE only takes 1.24 s to process 11 complete datasets (2^{11} data points) for 11 genes according to Gao et al. (2018), Mat_DNF takes 483.1 s, which suggests the need for improving implementation of Mat_DNF for example by parallel technologies.

Also we observe differences in terms of "score" which the number of genes whose domain (regulators) is correctly inferred when the learning data is not complete. We randomly sample m states and their state transitions and measure scores for $m = 80, 160$ by running Mat_DNF on sampled transitions.[18] We repeat this trial five times and take the average. The results are score = 8.8 for $m = 80$ and score = 10.6 for $m = 160$, which are lower than those by rfBFE reported in Gao et al. (2018) where score = 10.8 for $m = 80$ and score = 10.9 for $m = 160$ respectively. This may be due to the lack of a special mechanism in Mat_DNF to identify regulators (domain).

In the case of asynchronous learning data described above, Mat_DNF and rfBFE return Boolean formulas listed in Table 4.[19] Table 4 shows that Mat_DNF and rfBFE return exactly the same Boolean formulas except for gene PU.1 and both successfully recover six original Boolean formulas. Concerning PU.1 however, while Mat_DNF successfully recovers one of the two original disjuncts, rfBFE recovers no original disjunct or recovers only one of the four original conjuncts (assuming the original one is in CNF). So, as far as the target asynchronous BN (Krumsiek et al., 2011) is concerned, Mat_DNF seems qualitatively competitive with rfBFE, though learning is considerably slow.

[17] rfBFE is a combination of two algorithms, random forest for feature selection and the BestFit extension algorithm (Lähdesmäki et al., 2003) for Boolean formula discovery.

[18] Parameters are set to max_try = 10, max_itr = 1000, h = 10000 and over-iteration with extra_update = 20.

[19] The table format and Boolean formulas learned by rfBFE are borrowed from Gao et al. (2018). Fact denotes the original Boolean formulas. We run Mat_DNF with $\alpha = 0.005$, max_try = 10, max_itr = 1000, h = 4000 and over-iteration (extra_itr = 100).

7 Related work

From a logical point of view, Mat_DNF infers a matricized DNF as an interpolant by numerical optimization and there is no previous work of the same kind as far as we know. As Sect. 4 reveals, any interpolant represented by a matricized DNF $\varphi = (\mathbf{C}, \mathbf{D})$ between the positive and negative data is translated to a single layer ReLU network described by (2) with network parameters $(\mathbf{C}, \mathbf{D})$ and vice versa. This mutual translation is expected to contribute to cross-fertilization of NNs and logic. For example logical characterization of interpolants with good generalizability can contribute to designing NNs with high generalizability.

On the optimization side, our approach is categorized as continuous and unconstrained global optimization applied to DNFs instead of CNFs (Gu et al., 1996). What differs from traditional approaches surveyed in Gu et al. (1996) is the Mat_DNF's cost function, which for instance encodes a conjunction as a sum of piecewise multivariate linear terms unlike those in Gu et al. (1996) that encode a conjunction by a product of some functions in one form or another.

Representing Boolean formulas by matrix is an established idea. Theoretically we can represent any Boolean formula in n variables in terms of $2^n \times 2^n$ or $2n \times 2^n$ matrix (Cheng and Qi, 2010; Kobayashi and Hiraishi, 2014). Our matricized DNF representation also requires a matrix $\mathbf{C}$ of similar size, for example $2^{n-1} \times 2n$ to represent the n-parity function. The technique of learning and outputting Boolean formulas represented by matrix has already been applied to learning AND/OR BNs in Sato and Kojima (2021), but with different purposes. Sato and Kojima (2021) aims at finding useful logical patterns in the biological data whereas DNFs in this paper are learned to verify or suggest BNs.

Mat_DNF is a simple neuro-symbolic system that explicitly represents DNFs. From this neuro-symbolic viewpoint, we notice several NNs have been proposed that can learn DNFs (Towell and Shavlik, 1994; Payani and Fekri, 2019; Katzir et al., 2021). However, they all implicitly embed DNFs in their NN architecture. In KBANN-net (Towell and Shavlik, 1994), for example, a conjunction containing k literals is encoded as a neuron represented by a tree with k leaves, each having a link weight ω such as 4 for positive literal and $-\omega$ for negative one, and the neuron is activated when $k \cdot \omega$ exceeds bias $= (k - 1/2) \cdot \omega$. In Neural Logic Networks (Payani and Fekri, 2019), conjunctions are represented by a product of linear functions of the form $1 - m(1 - x)$ where $0 < m < 1$ and embedded in a neural network isomorphically to a DNF. In Net-DNF (Katzir et al., 2021), a trainable AND function is used: $\mathrm{AND}(\mathbf{x}) = \tanh((\mathbf{c} \bullet L(\mathbf{x})^T) - \|\mathbf{c}\|_1 + 1.5)$ where $L(\mathbf{x}) = \tanh(\mathbf{x}^T W + \mathbf{b})$ to encode conjunctions. As a result, they need an extra process to reconstruct a DNF from the learned parameters.

There are logical approaches to BN learning (Inoue et al., 2014; Tourret et al., 2017; Chevalier et al., 2019; Gao et al., 2022). Logically our work can be considered as a matricized version of "learning from interpretation transition" in logic programming in which a BN is represented by a propositional normal logic program (Inoue et al., 2014; Gao et al., 2022). The most related work is NN-LFIT proposed by Tourret et al. (2017) which performs two-stage DNF learning. First a single layer feed-forward NN is trained by state transition data. Then learned parameters irrelevant to the output are filtered out and DNFs are extracted from the remaining parameters. However since their performance evaluation is based on error rate of learned rules, not recovery rate of the learned DNFs like ours, direct comparison is difficult.

8 Conclusion

We proposed a simple feed-forward neural network Mat_DNF for the end-to-end learning of Boolean functions. It learns a Boolean function and outputs a matricized DNF realizing the target function. It searches for a DNF as a root of a non-negative cost function by minimizing the cost function to zero. We also established a new connection between neural learning and logical inference. We proved the equivalence between DNF learning by Mat_DNF and the inference of interpolants in logic between the positive and negative input data. We applied Mat_DNF to learning two synchronous BNs and one asynchronous BN from biological literature and empirically confirmed the effectiveness of our approach.

While doing so, we introduced "domain ratio" *dr* as an indicator of data scarcity and defined generalization w.r.t. *dr*. By examining the generalizability of DNFs learned from scarce data while varying *dr*, we discovered two operations, noise-expansion (expanding input vectors with noise vectors) and over-iteration (continuing learning after learning error reaches zero), can considerably improve generalizability by shifting the choice of a learned DNF. These two operations explain high recovery rate of original DNFs in our BN learning experiments.

Future work includes a reimplementation of Mat_DNF by GPUs, the refinement of noise-expansion and over-iteration and pursuing the idea of binary classifier as logical interpolant.

Author's contributions TS is a major contributor in writing the manuscript. KI assists in preparing the manuscript and financial support. All authors read and approved the final manuscript.

Funding This work is supported by JSPS KAKENHI Grant Number JP21H04905 and JST CREST Grant Number JPMJCR22D3.

Availability of data and material Not Applicable.

Code availability Mat_DNF is available upon request as an octave program.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethics approval Not Applicable.

Consent to participate Not Applicable.

Consent for publication Not Applicable.

References

Cheng, D., & Qi, H. (2010). A linear representation of dynamics of Boolean networks. *IEEE Transactions on Automatic Control, 55*(10), 2251–2258. https://doi.org/10.1109/TAC.2010.2043294

Chevalier, S., Froidevaux, C., Paulevé, L., & Zinovyev, A. (2019). Synthesis of boolean networks from biological dynamical constraints using answer-set programming. In: 31st International Conference on Tools with Artificial Intelligence. ICTAI

Craig, W. (1957). Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic, 22*(3), 269–285.

Fauré, A., Naldi, A., Chaouiya, C., & Thieffry, D. (2006). Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics, 22*(14), 124–131. https://doi.org/10.1093/bioinformatics/btl210

Feldman, V. (2007). Efficiency and computational limitations of learning algorithms. PhD thesis, USA. AAI3251269

Gao, S., Xiang, C., Sun, C., Qin, K., & Lee, T.H. (2018). Efficient Boolean Modeling of Gene Regulatory Networks via Random Forest Based Feature Selection and Best-Fit Extension. In: 2018 IEEE 14th International Conference on Control and Automation (ICCA), pp. 1076–1081 (2018). https://doi.org/10.1109/ICCA.2018.8444221

Gao, K., Wang, H., Cao, Y., & Inoue, K. (2022). Learning from interpretation transition using differentiable logic programming semantics. *Machine Learning, 111*(1), 123–145. https://doi.org/10.1007/s10994-021-06058-8

Gu, J., Purdom, P.W., Franco, J., & Wah, B.W. (1996). Algorithms for the satisfiability (sat) problem: A survey. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 19–152

Hansen, K. A., & Podolskii, V. V. (2015). Polynomial threshold functions and boolean threshold circuits. *Information and Computation, 240*, 56–73. https://doi.org/10.1016/j.ic.2014.09.008

Inoue, K., Ribeiro, T., & Sakama, C. (2014). Learning from Interpretation Transition. *Machine Learning, 94*(1), 51–79.

D.J.Irons: (2009). Logical analysis of the budding yeast cell cycle. Journal of theoretical biology *257*(4)

Ishida, T., Yamane, I., Sakai, T., Niu, G., & Sugiyama, M. (2020). Do we need zero training loss after achieving zero training error? CoRR, ICML2020 poster

Kamath, A. P., Karmarkar, N., Ramakrishnan, K. G., & Resende, M. G. C. (1992). A continuous approach to inductive inference. *Mathematical Programming, 57*, 215–238.

Katzir, L., Elidan, G., & El-Yaniv, R. (2021). Net-dnf: Effective deep modeling of tabular data. In: Proceedings of the 9th International Conference on Learning Representations (ICLR 2021)

Kauffman, S. (1969). Homeostasis and differentiation in random genetic control networks. *Nature, 51*, 177–178. https://doi.org/10.1038/224177a0

Kingma, D.P., & Ba, J. (2015). Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, (ICLR 2015) Conference Track Proceedings

Kobayashi, K., & Hiraishi, K. (2014). Ilp/smt-based method for design of Boolean networks based on singleton attractors. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 11*, 1253–1259.

Krumsiek, J., Marr, C., Schroeder, T., & Theis, F. (2011). Hierarchical differentiation of myeloid progenitors is encoded in the transcription factor network. *PloS One, 6*, 22649. https://doi.org/10.1371/journal.pone.0022649

Lähdesmäki, H., Shmulevich, I., & Yli-Harja, O. (2003). On learning gene regulatory networks under the boolean network model. *Machine Learning, 52*(1–2), 147–167. https://doi.org/10.1023/A:1023905711304

Liang, S., Fuhrman, S., & Somogyi, R. (1998). REVEAL, A General Reverse Engineering Algorithm for Inference of Genetic Network Architectures. In: Pacific Symposium on Biocomputing, vol. *3*, pp. 18–29 (1998)

Malach, E., & Shalev-Shwartz, S. (2019). Learning boolean circuits with neural networks. CoRR

McMillan, K.L., In: Clarke, E.M., Henzinger, T.A., Veith, H., & Bloem, R. (2018). (eds.) Interpolation and Model Checking, pp. 421–446. Springer, Cham

Mixon, D.G., & Peterson, J. (2015). Learning Boolean functions with concentrated spectra. In: Papadakis, M., Goyal, V.K., Ville, D.V.D. (eds.) Wavelets and Sparsity XVI, vol. 9597, pp. 88–95. SPIE, ??? (2015). International Society for Optics and Photonics. https://doi.org/10.1117/12.2189112

Oliveira, A.L., & Sangiovanni-Vincentelli, A. (1993). Learning complex boolean functions: Algorithms and applications. In: Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'93), pp. 911–918 (1993)

Payani, A., & Fekri, F. (2019). Learning algorithms via neural logic networks. CoRR **abs/1904.01554** (2019)

Power, A., Burda, Y., Edwards, H., Babuschkin, I., & Misra, V. (2021). Grokking: Generalization beyond overfitting on small algorithmic datasets. 1st Mathematical Reasoning in General Artificial Intelligence Workshop

Ribeiro, T., Folschette, M., Magnin, M., & Inoue, K. (2021). Learning any memory-less discrete semantics for dynamical systems represented by logic programs. *Machine Learning*. https://doi.org/10.1007/s10994-021-06105-4

Rückert, U., & Kramer, S. (2003). Stochastic local search in k-term DNF learning. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003), pp. 648–655 (2003)

Sato, T., & Kojima, R. (2021). Boolean network learning in vector spaces for genome-wide network analysis. In: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR2021), pp. 560–569 (2021). https://doi.org/10.24963/kr.2021/53

Sharma, R., Nori, A.V., & Aiken, A. (2012). Interpolants as classifiers. In: Computer Aided Verification, pp. 71–87. Springer, Berlin

Tourret, S., Gentet, E., & Inoue, K. (2017). Learning human-understandable description of dynamical systems from feed-forward neural networks. In: Advances in Neural Networks—14th International Symposium, Proceedings, Part I, LNCS 10261, pp. 483–492

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence, 70*(1), 119–165. https://doi.org/10.1016/0004-3702(94)90105-8

van Krieken, E., Acar, E., & van Harmelen, F. (2022). Analyzing differentiable fuzzy logic operators. *Artificial Intelligence, 302*, 103602. https://doi.org/10.1016/j.artint.2021.103602

Vizel, Y., Weissenbacher, G., & Malik, S. (2015). Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE, 103*(11), 2021–2035.

Zhang, Z., Zhao, Y., Liu, J., Wang, S., Tao, R., Xin, R., & Zhang, J. (2019). A general deep learning framework for network reconstruction and dynamics learning. Applied Network Science *4*(110)