Neural Disjunctive Normal Form: An Inherently Interpretable Deep Learning Classifier

Jialin Lu   Martin Ester

Abstract

We present Neural Disjunctive Normal Form (Neural DNF), an inherently interpretable deep learning classifier. Neural DNF uses a standard neural network as feature extractor to produce Boolean intermediate outputs which are then inputted to a set of IF-THEN rules for classification. We view interpretability as the essential merit of Neural DNF. By providing the rules as explanations, Neural DNF is inherently interpretable and needs no post-hoc interpretation. Compared with the commonly-used linear model, rules describe the decision boundary in an explicit and transparent way. End-to-end learning of the neural network and rule-based classifier is challenging, for which we propose the *BOAT* algorithm.

1 Introduction

For decades, the emphasis of machine learning research has been to increase the capacity to accurately model complex patterns, but typically such flexibility results in "blackbox models" that are too complex for humans to understand. As many blackbox deep learning models are deployed in real-world applications, post-hoc methods are often employed to probe the already-trained blackbox to derive explanations of the model and its predictions. Instead of making post-hoc interpretations, we consider to build inherently interpretable[1] classifiers, also known as self-explaining classifiers, in the sense that the interpretability is built-in architecturally and the explanation it derives for making a prediction is exactly how it computes the prediction.

A deep learning classifier f can essentially be viewed as a two-stage model $f = g \circ \phi$ whose prediction on a sample x is given by $\hat{y} = f(x) = g(\phi(x))$ where the first-stage ϕ is a neural network feature extractor and the second-stage g is the actual classifier. To build inherently interpretable deep learning classifiers, some recent works [29; 42; 8] adopt this two-stage paradigm and propose various ways for developing an *interpretable* first-stage feature extractor ϕ. While the first stage is highly customizable and leaves a lot of possibilities to explore per application, all these previous works choose a linear model for the second-stage classifier. This is not only because linear model is regarded as interpretable (its coefficients can be interpreted as feature importance), but also that a linear model g can be optimized jointly with the feature extractor ϕ by backpropagation.

In this paper, we argue that there are two major reasons for preferring rules over linear models as the second-stage classifier: (1) **Rules as combinations of conditions are more interpretable than feature importance** (at least for classification). It is widely acknowledged that the rule-based models are interpretable [14; 19; 43] because the rules give explanations by explicitly describing the decision boundary as logical combinations of conditions. Therefore, rules can more naturally provide counterfactual explanations in the form of *'Would changing a certain factor have changed the decision?'* which is often considered to be the most important property of explanation [13; 43; 16; 30]. On the other hand, coefficients as feature importances are arguably 'harder to use and to understand for a non-expert user' [43]. We conjecture that rules are probably closer to human's mental model for classification, which can be corroborated by the fact that rules have been the most intuitive choice of model for human categorization learning [6]. (2) **Using rules as a second-stage**

[1]For reasons on favoring inherently interpretable models over post-hoc interpretations, we refer interested readers to the inspiring discussion by Rudin [35].

Preprint. Under review.

classifier g requires the feature extractor ϕ to produce Boolean output($\{0, 1\}$), which is less complex than continuous ones. Compared with continuous values, Boolean output is simpler as it has only two states, making it easier for human to probe its meaning or to potentially align it with human knowledge.

As one step towards more interpretable deep learning models, we introduce Neural DNF, which uses rules as the second-stage classifier. By presenting the rules as explanations, Neural DNF is inherently interpretable in the sense the explanation of Neural DNF is exactly how Neural DNF computes the prediction, so the provided explanations are *faithful* by design. Despite many reasons for favoring rules over linear models, the major technical challenge for Neural DNF is model learning: end-to-end learning of rule-based module together with the feature extractor network is not directly possible. In order to do so, we propose BOAT (Bi-Optimizer learning with Adaptively-Temperatured noise), a two-optimizer approach to learn ϕ and g jointly (see fig. 1 for an overview): we use standard continuous-parameter Adam optimizer [25] to optimize the first-stage neural network ϕ and use a binary-parameter optimizer adopted based on [18] to optimize the binary parameters of the second-stage rule-based model g. As the novel key ingredient of BOAT, we propose adaptively-temperatured noise to perform weight perturbation which enables the learning of the rule-based g. Our experiments demonstrate that learning constantly fails without such noise.

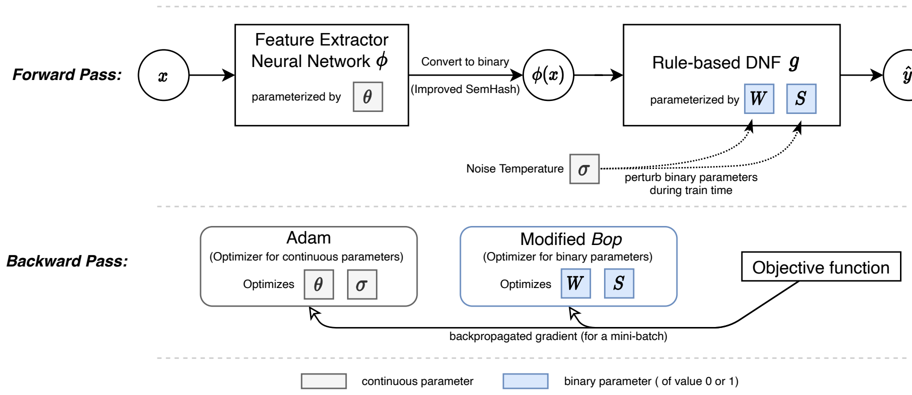

Forward Pass:
x
Feature Extractor Neural Network ϕ
parameterized by θ
Convert to binary
(Improved SemHash)
$\phi(x)$
Rule-based DNF g
parameterized by W S
$\hat{y}$
Noise Temperature σ
perturb binary parameters during train time
Backward Pass:
Adam
(Optimizer for continuous parameters)
Optimizes θ σ
Modified *Bop*
(Optimizer for binary parameters)
Optimizes W S
Objective function
backpropagated gradient (for a mini-batch)
continuous parameter
binary parameter (of value 0 or 1)

Figure 1: *Overview of Neural DNF and the proposed BOAT Algorithm*

2 Neural Disjunctive Normal Form

We consider a standard supervised learning setting of classification: given a dataset of $D = \{(x, y)\}_1^{|D|}$, we wish to learn a classification function f. For simplicity, we now assume y to be binary labels ($y \in \{0, 1\}$) and will extend to multi-class setting later. We adopt the aforementioned two-stage paradigm for building inherently interpretable classifiers: the classifier function $f = g \circ \phi$ is a composition of two functions ϕ and g, where ϕ is a neural network feature extractor, taking raw data x as input and returning a set of intermediate representations $\{c_1, c_2, \ldots, c_K\} \in \{0, 1\}^K$ where K is a predefined number. We use 0 and 1 to represent False and True. We call each c a concept *predicate* to emphasize that c has a Boolean interpretation: it indicates the presence or the absence of a concept. A rule-based module g, formulated as a Disjunctive Normal Form (DNF), is used for the actual classification. The overall classification is given by $\hat{y} = f(x) = g(\phi(x))$.

The neural network feature extractor ϕ processes the raw input x into a set of intermediate representations $\{c_1, c_2, \ldots, c_k\}$ called concept predicates. As ϕ is parameterized by a neural network θ, ϕ's output $\tilde{c_i}$ is real-valued. We use a binary step function to discretize $\tilde{c_i}$ into Boolean predicates: $c_i = 1$ if $\tilde{c_i} > 0$ and $c_i = 0$ otherwise. However, since gradient through this step function is zero almost anywhere and thus prevents training, we utilize the *Improved SemHash*[21]. It does not need any manual annealing of temperature [7; 17] or additional loss functions and has been demonstrated to be a robust discretization technique in various domains [21; 10; 22].

Interpretability desiderata of ϕ are highly subjective and are the focus of the aforementioned previous works on interpretable deep learning. Ideally, the intermediate output $\boldsymbol{c}$ should be the high-level 'natural' features that are aligned with human-comprehensible concepts. As an example from Melis and Jaakkola [29], for medical image processing, the concept predicate $\boldsymbol{c}$ should indicate tissue ruggedness, irregularity, elongation, etc, and are 'the first aspects that doctors look for when diagnosing'. However, this is ill-defined and more often there is no prior knowledgebase to do so, in which case some techniques can be employed. For example, we can enforce $\boldsymbol{c}$ to represent diverse aspects of the original data by some unsupervised representation learning[29], or we can bind each c_i with some 'prototypes' [8]. Practically, we emphasize that ϕ is domain-specific and highly customizable. From this perspective, the design of ϕ is not the focus of this paper. For simplicity, we choose not to elaborate on ϕ and only assume ϕ to be some generic feedforward neural network.

The rule-based classifier g is formulated as a Disjunctive Normal Form (DNF). g takes a set of Boolean predicates as input feature and makes binary prediction. g can be more intuitively interpreted as a decision set, consisting of a set of if-then rules: g predicts the positive class if at least one of the rules is satisfied and predicts the negative class otherwise. We define a rule r_i to be a conjunction of one or more conditions (literals): $r_i = b_{i1} \wedge b_{i2} \wedge \ldots$ where b_j for $j \in \{1, 2, \cdots 2K\}$ can be a predicate c or its negation $\neg c$. A DNF is a disjunction of one or more rules: $r_1 \vee r_2 \vee \ldots \vee r_n$.

Rule learning algorithm for DNF can be viewed as a subset selection problem: first, a pool of all possible candidate rules is constructed and then a learning algorithm is applied to find a good subset of rules as the learned DNF. Let N to be the size of pool of rules and K the number of predicates as input, we now formulate using a binary matrix $\boldsymbol{W}_{2K\times N}$ and a binary vector $\boldsymbol{S}_N$. $\boldsymbol{W}_{2K\times N}$ represents the pool of candidate rules such that each column represents a rule and the non-zero elements of a column indicate the conditions of that rule. $\boldsymbol{S}_N$ can be viewed as a membership vector that determines which rules are selected as the learned DNF. Given the input concept predicates $\boldsymbol{c} = \{c\}_1^K$, g computes the Boolean function as:

$$\hat{y} = g(\boldsymbol{c}) = \bigvee_{\boldsymbol{S}_j=1}^{N} \bigwedge_{\boldsymbol{W}_{i,j}=1} b_i \qquad \text{where } \{b\}_i^{2K} = \{c_1, \neg c_1, c_2, \neg c_2, \ldots, c_k, \neg c_k\} \tag{1}$$

Note that eq. (1) is used during training; after training, the original pool of rules can be discarded and only the selected rules are stored. Conventional methods of rule learning(we refer [27; 44] as some recent works) usually construct a fixed $\boldsymbol{W}$ by rule pre-mining; and then 'learning' corresponds to the discrete optimization of the membership vector $\boldsymbol{S}$. However, this approach is not compatible if we wish to jointly optimize g with the neural network ϕ end-to-end.
In order to do so, we introduce two essential modifications: (1) we make both $\boldsymbol{W}_{2K\times N}$ and $\boldsymbol{S}_N$ learnable parameters; (2) we introduce a differentiable replacement[2] of the logical operation of eq. (1) so that gradients can be properly backpropagated:

$$r_j = \bigwedge_{\boldsymbol{W}_{i,j}=1} b_i \quad \xrightarrow{\text{replaced by}} \quad r_j = \prod_i^{2k} \boldsymbol{F}_{\text{conj}}(b_i, \boldsymbol{W}_{i,j}), \text{ where } \boldsymbol{F}_{\text{conj}}(b, w) = 1 - w(1-b) \tag{2}$$

$$\hat{y} = \bigvee_{\boldsymbol{S}_j=1} r_j \quad \xrightarrow{\text{replaced by}} \quad \hat{y} = 1 - \prod_j^{N}(1 - \boldsymbol{F}_{\text{disj}}(r_j, \boldsymbol{S}_j)), \text{ where } \boldsymbol{F}_{\text{disj}}(r, s) = r \cdot s \tag{3}$$

eqs. (2) and (3) computes the DNF function exactly as eq. (1). However, note that $\boldsymbol{W}$ and $\boldsymbol{S}$ are binary, optimizing $\{\boldsymbol{W}, \boldsymbol{S}\}$ with mini-batch gradient descent is non-trivial. The above formulation can be easily extended to multi-class settings by using a different $\boldsymbol{S}$ for each class; in test time when more than one classes are predicted as positive, some tie-breaking procedure can be employed. In this paper, we simply use lazy tie-breaking by selecting the first encountered positive class in ascending order (e.g. when class 1, 4, 7 are all predicted as positive, we select class 1).

Interpretability desiderata of g can be measured as the compactness of the DNF. Here we consider the simplest case: we want the total number of rules and the length of rules to be small. Note that as only rules selected by $\boldsymbol{S}$ are actually used, we use a grouped L1-norm by $\mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S}) = \lambda_g \sum_j^N |\boldsymbol{S}_j|_1 |\boldsymbol{W}_{\cdot,j}|_1$ and leave more sophisticated metrics like feature overlaps [27] as future work.

Objective function of Neural DNF is given as $\boldsymbol{L} = \mathbb{L}_{loss} + \lambda_g \mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S})$ where $\mathbb{L}_{loss} = \frac{1}{|D|}\sum_{(x,y)\in D} L(y, g_{\boldsymbol{W}}(\phi_\theta(x)))$ is mean squared error and $\mathbb{R}_g$ being the regularization terms for g defined above.

[2]Obtaining eqs. (2) and (3) is not new, similar formulation can be found in the literature such as the logical activation functions [33; 45] or soft NAND gate [36].

3 The *BOAT* algorithm for learning Neural DNF

In this section, we introduce Bi-Optimizer learning with Adaptively-Temperatured noise(BOAT), the learning algorithm for Neural DNF. BOAT utilize using two optimizers, treating binary and continuous parameters separately: a standard deep learning optimizer Adam [25] is used to optimize the continuous parameters θ of the neural network ϕ and a binary-parameter optimizer adopted based on [Helwegen et al., 2019] is used to optimize the binary parameters $\{\boldsymbol{W}, \boldsymbol{S}\}$ of the DNF module g. As the key ingredient of BOAT, during train time the binary parameters $\boldsymbol{W}$ and $\boldsymbol{S}$ are perturbed by noise whose magnitude is controlled by the noise temperature σ (eq. (4)). We emphasize that (1) the introduced noise is necessary as otherwise learning of $\{\boldsymbol{W}, \boldsymbol{S}\}$ constantly fails even in very simple cases (see section 4.3) and (2) the noise temperature σ is also a learnable continuous parameter, which can be optimized together with other continuous parameters by Adam.

Joint optimization of $g_{\boldsymbol{W},\boldsymbol{S}}$ and ϕ_θ is non-trivial, because although the overall model is differentiable, $\{\boldsymbol{W}, \boldsymbol{S}\}$ consists of binary values ($\{0, 1\}$) and thus standard deep learning optimizers designed for continuous parameters cannot be directly applied. One alternative, denoted as DNF-Real [33; 45], is to simply use real-valued weight $\{\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}\}$ transformed using sigmoid/tanh functions as a surrogate and then use Adam as the optimizer. After training, real-valued weights are thresholded to binary values, and performance drop can occur. In practice, we find DNF-Real to be optimization-friendly just like any other real-valued DNNs but it is not guaranteed to result in binary-valued parameters. Another alternative, which we denote as DNF-STE, is adopted from binary neural network research[11; 12]. It maintains real-valued *latent* parameters which are binarized in forward pass computation. In backward computation the gradients are updated to the latent real-valued parameters using the straight-through estimator (STE) [3]. This technique is widely used and works quite well for large-scale binary neural networks. However, for a small-sized DNF g, DNF-STE is very sensitive to initialization and hyperparameters and can easily be stuck in local minima.

BOAT Algorithm. We introduce the BOAT algorithm, which combines the best of both of the approaches discussed above: (1) It is optimization-friendly and not very sensitive to initialization and hyperparameters. This is especially important as g (the second stage of Neural DNF) is typically not overparameterized; (2) It naturally optimizes binary-valued parameters. BOAT uses a modified version of the *Bop* optimizer [18] to optimize the binary-valued parameter $\{\boldsymbol{W}, \boldsymbol{S}\}$ given gradient as learning signals. We make minor modifications to suit *Bop* into the case of $\{0, 1\}$. The Modified *Bop* optimizer introduced in this paper uses gradient as the learning signal and flips the value of $w \in \{0, 1\}$ only if the gradient signal m exceeds a predefined *accepting threshold* τ:

$$w = \begin{cases} 1 - w, & \text{if } |m| > \tau \text{ and } (w = 1 \text{ and } m > 0 \text{ or } w = 0 \text{ and } m < 0) \\ w, & \text{otherwise.} \end{cases}$$

where m is the gradient-based learning signal computed in the backward pass. A non-zero τ is introduced to avoid rapid back-and-forth flips of the binary parameter and we find it helpful to stabilize the learning because m is of high variance. To obtain consistent learning signals, instead of using vanilla gradient ∇ as m, the exponential moving average of gradients is used

$$m = \gamma m + (1 - \gamma)\nabla$$

where γ is the exponential decay rate and ∇ is the gradient computed for a mini-batch. We use $\gamma = 1 - 10^{-5}$ and $\tau = 10^{-6}$ throughout this paper while tuning the configuration of γ and τ can be founded in the original *Bop* paper [18].

Adaptively-temperatured Noise. The reason we introduce adaptively-temperatured noise is that simply using the introduced Modified *Bop* shares the same drawback as using DNF-STE: optimization is very sensitive to initialization and hyperparameters and can be stuck in local minima very easily. When stuck in local minima, the gradients w.r.t $\boldsymbol{W}$ and $\boldsymbol{S}$ effectively become zero, and thus any further updates for $\boldsymbol{W}$ and $\boldsymbol{S}$ are disabled. We suspect the reason is that even the DNF function eqs. (2) and (3) is well defined on $[0, 1]$, since the choice of values of $\boldsymbol{W}$ and $\boldsymbol{S}$ can only take $\{0, 1\}$, the loss surface is non-smooth and thus the optimization becomes hard. To overcome this, we propose to perturb the binary weights w during training by adding noise in the forward pass such that the perturbed $\tilde{w}$ lies in $[0, 1]$. We believe this introduced noise smoothes the loss surface and helps the optimization of $\boldsymbol{W}, \boldsymbol{S}$. Specifically, for every entry w in $\boldsymbol{W}$ and $\boldsymbol{S}$, we utilize a noise temperature parameter $\sigma_w \in [0, 0.5]$ to perturb w with noise as follows:

$$\tilde{w} = \begin{cases} 1 - \sigma_w \cdot \epsilon & \text{if } w = 1 \\ 0 + \sigma_w \cdot \epsilon & \text{if } w = 0 \end{cases}, \text{ where } \epsilon \sim \text{Uniform}(0, 1) \tag{4}$$

Algorithm 1 The BOAT algorithm for learning Neural DNF

Hyperparameters: Accepting threshold $\tau > 0$; Exponential decay rate $\gamma \in [0, 1)$; Initial noise temperature $\sigma_0 \in [0, 0.5]$; Size of rule pool N. (Default:τ=10^{-6}, γ=1-10^{-5}, σ_0=0.2, N=64.)
Input: Dataset $\boldsymbol{D}$; **Output**: The DNF $g\left(\{\boldsymbol{W}, \boldsymbol{S}\}\right)$; The neural network $\phi\left(\theta\right)$.

```
 1: Initialize $\theta$ randomly.
 2: For every $w$ in $\{\boldsymbol{W}, \boldsymbol{S}\}$: initialize $w \in \{0, 1\}$ randomly, $m_w \leftarrow 0, \sigma_w \leftarrow \sigma_0$.
 3: while stopping criterion not met do
 4:     Sample mini-batch $\{(x_i, y_i)\}^{\text{batch size}}$ from the training set $\boldsymbol{D}$.
 5:     Compute the perturbed $\{\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}\}$ where each entry $\tilde{w}$ is perturbed according to $\sigma_w$ (eq. (4)).
 6:     Use $\theta$ and perturbed $\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}$ to compute the objective function $\sum_{x_i, y_i} \boldsymbol{L}(g_{\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}}(\phi_\theta(x_i)), y_i)$
 7:     for every binary parameter $w$ in $\{\boldsymbol{W}, \boldsymbol{S}\}$ do
 8:         Compute gradient $\nabla_w$ w.r.t the objective function computed at line 4.
 9:         Update exponential moving average of gradient: $m_w \leftarrow \gamma m_w + (1-\gamma)\nabla_w$.
10:         if $|m_w| > \tau$ and ($m_w < 0$ and $w = 0$ or $m_w > 0$ and $w = 1$) then
11:             $w \leftarrow 1 - w$        ▷ Line 7-11: update binary parameters using Modified Bop
12:     Update $\theta$ by Adam given the objective function computed at line 6.
13:     for every $\sigma_w$ do
14:         Update $\sigma_w$ by Adam given the objective function computed at line 6.
15:         Clip $\sigma_w = min(0.5, max(\sigma_w, 0))$
```

In training time the perturbed weight $\tilde{w}$ is used in the forward pass computation of the objective function; in test time, we simply disable this perturbation. In order to force σ_w lies in range $[0, 0.5]$, we clip σ_w by $\sigma_w = min(0.5, max(\sigma_w, 0))$ after evey mini-batch update. Note that σ_w is not globally shared: we have a σ_w for every w in $\boldsymbol{W}$ and $\boldsymbol{S}$ (so in total $2K * N + N$). We call it *Adaptively*-temperatured noise because we make σ_w also a learnable parameter. We simply initialize $\sigma_w = \sigma_0$ and optimize σ_w by Adam as well. The choice of initial value σ_0 requires heuristic: σ_0 cannot be too large as optimization will be slower (fig. 5c), and σ cannot be too small as in the extreme case, with zero noise the optimization of $\boldsymbol{W}, \boldsymbol{S}$ will constantly fail (fig. 5b). We find values between 0.1 and 0.3 all work fine and we use $\sigma_0 = 0.2$ as the default initial value.
Remark: We can also apply adaptively-temperatured noise for DNF-STE, but it converges slower (fig. 6). We conjecture the reason to be the acceptance threshold τ which effectively filters out noisy learning signals so that rapid flipping is prevented, suggested as the main advantage of *Bop* [18] .

4 Experiments

4.1 On a 2D toy dataset.

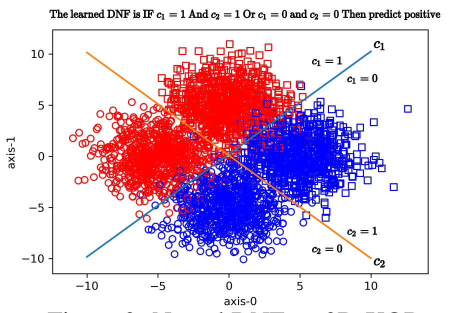


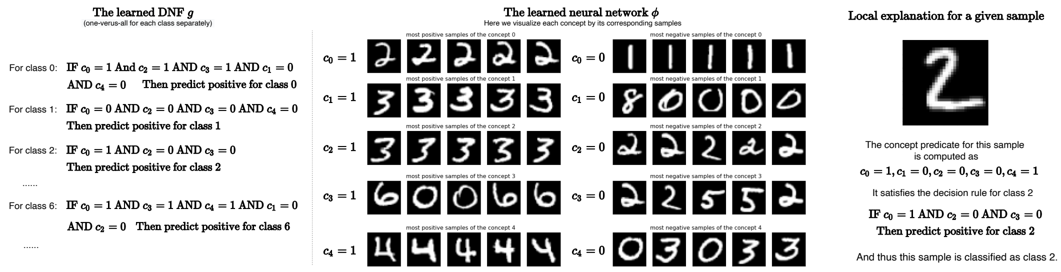
Figure 2: Neural DNF on 2D-XOR

Here we first apply Neural DNF on a 2D toy dataset 2D-XOR as it is straightforward to visualize (fig. 2). 2D-XOR is generated by first drawing four isotropic 2D Gaussian clusters and mark them as red square, blue square, blue circle and red circle (in clockwise order). We use a XOR-like label assigningment, labeling red squares and blue circles as positive and the rest two clusters as negative. For easy visualization, we use a one fully-connected layer network as ϕ to produce two concept predicates c_1, c_2. We expect the learned Neural DNF to find the two correct concept predicates that represent shape and color, respectively, and the correct classification function ('color is red and shape is square or color is not red and shape is not square'). As shown in fig. 2, we visualize the decision boundary of c_1 and c_2 by two lines and we find c_1, c_2 do separate red-against-blue and square-against-circle as expected. We view fig. 2 as a minimal working example of Neural DNF demonstrating that it is useful when the underlying classification process consists of subtypes (each described as one rule) and requires some logical compositions.

4.2 On image datasets.

In this section, we first give an anecdotal example on the explanations Neural DNF can derive (fig. 3) and then demonstrate empirically that given the same first-stage feature extractor, Neural DNF sacrifices slight loss of accuracy (table 1) while gaining more faithful interpretations (fig. 4).

In order to give an example of Neural DNF 's explanation, we first apply Neural DNF on MNIST as a direct comparison to the explanations provided by linear models (e.g., the MNIST example from [29]). We use a randomly-initialized LeNet-like convolutional network as the feature extractor ϕ to produce 5 concept predicates. Since MNIST has 10 class, we use a separate S for each class. After training Neural DNF finds one or two rules for each class and can achieve over 99% test accuracy.


The learned DNF g
(one-verus-all for each class separately)
For class 0: IF $c_0 = 1$ And $c_2 = 1$ AND $c_3 = 1$ AND $c_1 = 0$ AND $c_4 = 0$ Then predict positive for class 0
For class 1: IF $c_0 = 0$ AND $c_2 = 0$ AND $c_3 = 0$ AND $c_4 = 0$ Then predict positive for class 1
For class 2: IF $c_0 = 1$ AND $c_2 = 0$ AND $c_3 = 0$ Then predict positive for class 2
......
For class 6: IF $c_0 = 1$ AND $c_3 = 1$ AND $c_4 = 1$ AND $c_1 = 0$ AND $c_2 = 0$ Then predict positive for class 6
......
The learned neural network ϕ
Here we visualize each concept by its corresponding samples
most positive samples of the concept 0
most negative samples of the concept 0
$c_0 = 1$
$c_0 = 0$
most positive samples of the concept 1
most negative samples of the concept 1
$c_1 = 1$
$c_1 = 0$
most positive samples of the concept 2
most negative samples of the concept 2
$c_2 = 1$
$c_2 = 0$
most positive samples of the concept 3
most negative samples of the concept 3
$c_3 = 1$
$c_3 = 0$
most positive samples of the concept 4
most negative samples of the concept 4
$c_4 = 1$
$c_4 = 0$
Local explanation for a given sample
The concept predicate for this sample is computed as
$c_0 = 1, c_1 = 0, c_2 = 0, c_3 = 0, c_4 = 1$
It satisfies the decision rule for class 2
IF $c_0 = 1$ AND $c_2 = 0$ AND $c_3 = 0$
Then predict positive for class 2
And thus this sample is classified as class 2.

(a) Global explanation for the overall Neural DNF Model (b) Local explanation

Figure 3: Explanations provided by Neural DNF on MNIST

Here we provide the explanations that Neural DNF derives: (1) In fig. 3a, we provide a *global* explanation for Neural DNF that explains the working of the overall model, by presenting the rules for each class as well as the image samples that triggers each concept predicate ($c = 1$) or not ($c = 0$). As it is necessary to understand what each each concept c means, we provide representative samples for $c = 1$ and $c = 0$ by retrieving samples that maximize or minimize the pre-binarization value $\tilde{c}$. (2) In fig. 3b, Neural DNF is also able to derive *local* explanations which explain the classification for a particular sample. Given a test sample, we present the value of its concept predicate (computed by ϕ) as well as the satisfied rule by which prediction is computed. Note in particular that the satisfied rules are not only the explanation but also how the predication is exactly computed, thus *inherently* interpretable.

Unlike explaining by feature importance that can only indicate some of $\phi(x)$ contributes more significantly to the final prediction, Neural DNF's explanation describes the decision boundary explicitly as combinations of conditions. For example, in fig. 3b as the decision rule for class 2 is 'If c_0=1 and c_2=0 and c_3=0 Then class 2', it becomes clear that only c_0, c_2, c_3 are essential for predicting class 2 while c_1, c_4 are not. In other words, for this test sample, we know precisely that if we change any of c_1 or c_4, the prediction will remain but changing of any of c_0, c_2, c_3 will give a different prediction.

Table 1: Test Accuracy on some image datasets

	MNIST	KMNIST	SVHN	CIFAR10
Neural DNF	99.08%	95.43%	90.13%	67.91%
feedforward DNN	99.12%	95.86%	90.74%	70.43%
feedforward DNN (without Binarization)	99.11%	96.02%	91.45%	71.45%
SENN	98.48%	92.64%	90.79%	71.09%
SENN (without Binarization)	98.50%	91.46%	92.15%	72.32%

Having demonstrated that Neural DNF is more interpretable than methods using a linear second-stage classifier, we now analyze whether and how much accuracy Neural DNF loses compared to non-rule-based models. The models we compare are (1) Neural DNF which uses DNF rules as g, (2) a feedforward DNN (using a linear model as g) and (3) self-explaining neural network (SENN) [29] which uses a neural network to generate the coefficients of a linear model conditioned on the input. We use the same architecture for ϕ for all compared models for a fair comparison. Since Neural DNF needs to binarize the extracted features using Improved SemHash, which is not required for DNN or SENN, we also evaluate the case of DNN and SENN with and without binarization. As shown in table 1 we observe that all models including Neural DNF perform similarly well in terms of test accuracy across datasets, while Neural DNF does lose accuracy, but only slightly. By comparing the accuracy of DNN and SENN with/out binarization, we can see that both DNN and SENN loses accuracy slightly if the binarization is applied. So the loss of Neural DNF 's accuracy can be explained from two sources: (1) the binarization (Improved SemHash) and (2) the use of rules as the classifier instead of the more well-studied linear (additive) models. However, this should not be viewed as a weakness as long as the accuracy loss is not significant. We also believe that this issue can be alleviated by using a stronger ϕ: given a sufficiently flexible ϕ, the accuracy loss of Neural DNF should become negligible, like the case of simple dataset such as MNIST.

In the last series of experiments on the image datasets, we analyze the faithfulness of explanations: the explanation for a particular prediction should be faithful to the underlying computation of prediction. We adopt the faithfulness metric proposed by [29] which measures the correlation of the change of the prediction (class probabilities) and the change of the explanation (relevance scores of features) upon perturbation of test examples.[3] We report the faithfulness metric on test set for DNN, SENN and Neural DNF in fig. 4. For DNN, the simplest way to compute relevance scores is to use the coefficients of its final linear layer. Alternatively, for DNN, we can also utilize post-hoc interpretation methods (also known as attribution methods) to assign relevance scores for DNN. We evaluate some representative post-hoc methods: Guided Backprop (GB) [39], gradient SHAP [28], Integrated Gradient (IG) [41] and DeepLIFT [37]. For SENN, we use its self-generated coefficients as relevance scores as suggested in [29]. For Neural DNF, since assigning relevance score for rule-based model is often considered as straightforward and omitted in literature, we adopt the prediction difference analysis [34], a model-agnostic relevance score assignment method. It deletes information on the features and observes its according influence as relevance scores.

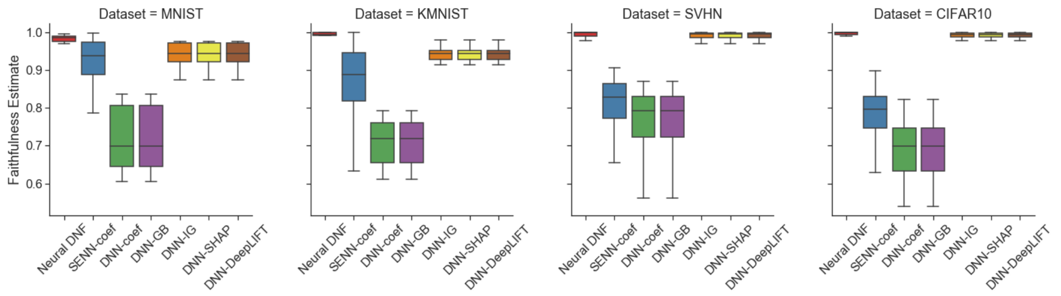


Figure 4: Evaluating the faithfulness of explanations on test set

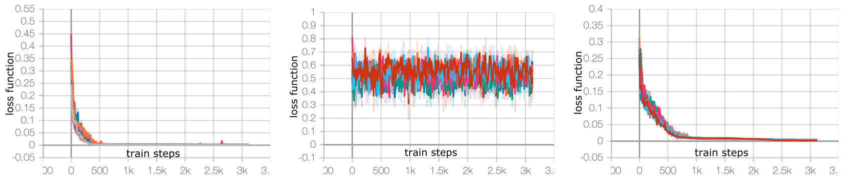
As shown in fig. 4, Neural DNF consistently achieves the highest faithfulness estimates for all datasets and its faithfulness has the smallest variance. This means Neural DNF's explanations are not just faithful, but faithful in a robust way across different test samples. For linear models that use the coefficients as relevance scores (see SENN-coef and DNN-coef in fig. 4), we find that SENN is more faithful than DNN. For DNN with post-hoc interpretation methods, while GB performs very similarly with DNN-coef, IG/SHAP/DeepLIFT drastically improves the faithfulness. This is because the latter three post-hoc methods compute the relevance w.r.t to a baseline reference, a concept that is now considered an essential improvement in post-hoc interpretation [40]. However, we pointed out that even these sophisticated post-hoc methods do not reach the same level of faithfulness as Neural DNF, in particular not on MNIST and KMNIST. We believe that Neural DNF's extremely high faithfulness is the direct result of the explicit decision boundary offered by rule-based models.

4.3 Evaluating BOAT for learning the DNF alone

Here we evaluate BOAT on datasets with Boolean features so that we learn only the DNF g. We use a synthetic dataset adopted from Payani and Fekri [33] which consists of 10-bit Boolean strings and a randomly-generated DNF as ground-truth.

First, to show the proposed noise is indeed necessary and helps the optimization, we learn the DNF with/without the noise on multiple datasets generated with different random seeds (so the ground-truth DNF is different) and plot the loss curves. From fig. 5a we observe that multiple runs of BOAT give very similar and stable convergence, while in fig. 5b, surprisingly, runs consistently fail to converge without the noise. We view this as strong evidence for the necessity of the proposed noise. On the other hand, if noise temperature is initialized larger, the convergence is slower (fig. 5b).

[3]As in [29], the relevance score assignment and feature perturbation is done for the extracted feature $\phi(x)$, not at the level of the raw data x.



(a) with noise (σ_0=0.2) (b) with NO noise (c) with noise (σ_0=0.4)

Figure 5: Loss curve with 10 differently-generated synthetic dataset

Next, we apply BOAT and compared the convergence speed with the following baselines: (1) DNF-Real [33; 45] and (2) DNF-STE which are discussed in section 3. (3) A Linear Model. (4) A multi-layer perceptron. Note that we also use the adaptive noise for DNF-STE since otherwise optimization will fail. As shown in fig. 6, BOAT gives the fastest convergence, while MLP, DNF-STE and DNF-Real converge much more slowly. The linear model does not converge. As for the reason of DNF-STE's slower and less stable convergence, we believe it is because that unlike the modified *Bop*, DNF-STE does not have the mechanism to prevent rapid flipping and thus optimization becomes more unstable. As for performance, all above methods except linear model achieve 100% F1 score on test set and BOAT can always find the ground truth DNF on different synthetic datasets generated with different random seeds. **Remark:** Note that since we use random initialization for $\boldsymbol{W}, \boldsymbol{S}$, a natural suspicion is that the ground truth DNF happens to be discovered by the random initialization, not learned. To refute it, we can use zero initialization and find that BOAT converges similarly to random initialization. We report it in the appendix together with the performance results on some UCI datasets.

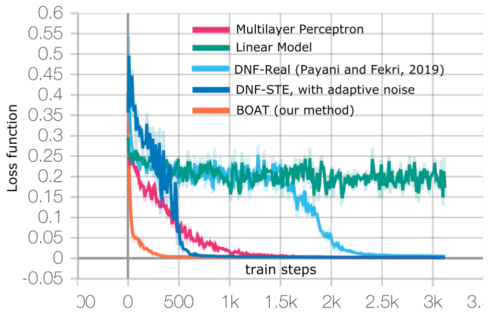


Figure 6: Loss Curve on the synthetic dataset

5 Conclusion and future work

We present Neural DNF as one step towards the development of an inherently interpretable deep learning classifier. Neural DNF also contributes to the fundamental problem of integrating continuous perception and logical reasoning (neural-symbolic integration [4; 15]). Our approach differs with previous works by using rules as the second-stage classifier, which provides extra interpretability, and we propose the BOAT algorithm for joint learning of rules and the feature extractor neural network. One re-occuring theme of the techniques we used in this paper is that we wish to avoid tedious manual tuning, e.g. we use Improved SemHash to binarize extracted features because it does not need to manually tune any extra parameters or add any additional losses; we propose the adaptive temperatured-noise because it is a minimal modification to help learning but again the noise temperature does not need manual tuning and can be also optimized.

In this paper, we focus only on the second stage of the deep learning classifier. However, if the extracted features from the first stage are non-interpretable, i.e. if they do not correspond to meaningful concepts, the explanation provided by the second stage becomes non-interpretable as well. Therefore, it is important for future works to develop an interpretable feature extractor. But this is a hard problem in two aspects: (1) To obtain interpretable features that identify human-comprehensible concepts, we require very rich and detailed knowledge such as representation learning techniques [9] that disentangles concepts, robust training against noise [23] or extra annotations [26; 2]. And any approach should be highly domain-specific for differerent data types and tasks [8; 5; 24; 20]. (2) For features to be *interpretable*, we literally need to properly *interpret* the true meaning of extracted features [38; 46; 32; 31]. This is a difficult task and is inevitably vulnerable to confirmation biases, which have been pointed out as a critical issue of interpretability research [1].

Another direction for future research is the use of rule languages that are more powerful than DNF (propositional logic). Since BOAT requires only minimal changes of the standard deep learning optimization, we believe it can be potentially applied to models with a more powerful second-stage rule-based module, or to other models with mixed binary-continuous parameters.

Broader Impact

Positive impact. As interpretability or explainability has become important in recent years [43], applying Neural DNF as an interpretable deep learning classifier to some domains is a natural extension of our work. Ideally, it will reduce the gap in stakeholders' understanding of AI systems and help alleviate many trust/accountability/responsibility-related issues in critical tasks and high-risk domains.

Negative impact. However, we are aware that using an interpretable model like Neural DNF does not solve every problem because very often, interpretability is not the end goal itself, but only a means towards some end goals like safety and fairness. For Neural DNF to make decisions/predictions in critical tasks or high-risk domains, we mention two necessary issues which if not carefully examined might bring unexpected risks in decision-making: (1) Neural DNF is an interpretable model but be aware that this interpretability is built on the interpretable features that are extracted from raw data. To ensure the overall model does not go wrong, we must first make sure the extracted features are interpretable, robust and stable; otherwise risks can still occur. But in general, current deep learning models as feature extractors are very vulnerable to adverserial attacks and an interpretable Neural DNF does not solve this problem. (2) As the space of Neural DNF's extracted features (Boolean concept predicates) and that of Neural DNF's rules are of finite size, model verification becomes easier than for continuous-valued features. We do not discuss verification in this paper, but when deployed in high-risk domains, we believe model verification will be very important for safety and security.

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9505–9515, 2018.

[2] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

[5] Carlo Biffi, Ozan Oktay, Giacomo Tarroni, Wenjia Bai, Antonio De Marvao, Georgia Doumou, Martin Rajchl, Reem Bedair, Sanjay Prasad, Stuart Cook, et al. Learning interpretable anatomical features through deep generative models: Application to cardiac remodeling. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 464–471. Springer, 2018.

[6] Jerome S Bruner, Jacqueline J Goodnow, and George A Austin. A study of thinking. 1956.

[7] Adrian Bulat, Georgios Tzimiropoulos, Jean Kossaifi, and Maja Pantic. Improved training of binary networks for human pose estimation and image recognition. *arXiv preprint arXiv:1904.05868*, 2019.

[8] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, pages 8928–8939, 2019.

[9] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[10] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9172–9180, 2019.

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[12] Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Bnn+: Improved binary network training. *arXiv preprint arXiv:1812.11800*, 2018.

[13] Finale Doshi-Velez, Mason Kortz, Ryan Budish, Chris Bavitz, Sam Gershman, David O'Brien, Stuart Schieber, James Waldo, David Weinberger, and Alexandra Wood. Accountability of ai under the law: The role of explanation. *arXiv preprint arXiv:1711.01134*, 2017.

[14] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[15] A Garcez, M Gori, LC Lamb, L Serafini, M Spranger, and SN Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4), 2019.

[16] Rory Mc Grath, Luca Costabello, Chan Le Van, Paul Sweeney, Farbod Kamiab, Zhao Shen, and Freddy Lécué. Interpretable credit application predictions with counterfactual explanations. *CoRR*, abs/1811.05245, 2018. URL `http://arxiv.org/abs/1811.05245`.

[17] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. Unsupervised neural generative semantic hashing. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 735–744, 2019.

[18] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In *Advances in neural information processing systems*, pages 7531–7542, 2019.

[19] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[21] Łukasz Kaiser and Samy Bengio. Discrete autoencoders for sequence models. *arXiv preprint arXiv:1801.09797*, 2018.

[22] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[23] Ashkan Khakzar, Shadi Albarqouni, and Nassir Navab. Learning interpretable features via adversarially robust optimization. *CoRR*, abs/1905.03767, 2019. URL `http://arxiv.org/abs/1905.03767`.

[24] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, pages 2942–2950, 2017.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015.

[27] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1675–1684, 2016.

[28] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[29] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, pages 7775–7784, 2018.

[30] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[31] Anh Nguyen, Jason Yosinski, and Jeff Clune. Understanding neural networks via feature visualization: A survey. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 55–76. Springer, 2019.

[32] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

[33] Ali Payani and Faramarz Fekri. Learning algorithms via neural logic networks. *arXiv preprint arXiv:1904.01554*, 2019.

[34] Marko Robnik-Šikonja and Igor Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008.

[35] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[36] Mehdi Sajjadi, Mojtaba Seyedhosseini, and Tolga Tasdizen. Disjunctive normal networks. *Neurocomputing*, 218:276–285, 2016.

[37] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[38] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[39] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[40] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Visualizing the impact of feature attribution baselines. *Distill*, 5(1):e22, 2020.

[41] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[42] Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N Nair. Explainable neural networks based on additive index models. *arXiv preprint arXiv:1806.01933*, 2018.

[43] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harvard Journal of Law & Technology*, 31(2), 2018.

[44] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. A bayesian framework for learning rule sets for interpretable classification. *The Journal of Machine Learning Research*, 18(1):2357–2393, 2017.

[45] Zhuo Wang, Wei Zhang, Nannan Liu, and Jianyong Wang. Transparent classification with multilayer logical perceptrons and random binarization. *ArXiv*, abs/1912.04695, 2019.

[46] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.