NEURAL DISJUNCTIVE NORMAL FORM: VERTICALLY INTEGRATING LOGIC WITH DEEP LEARNING FOR CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present Neural Disjunctive Normal Form (Neural DNF), a hybrid neuro-symbolic classifier that *vertically* integrates propositional logic with a deep neural network. Here, we aim at a *vertical* integration of logic and deep learning: we utilize the ability of deep neural networks as feature extractors to extract intermediate representation from data, and then a Disjunctive Normal Form (DNF) module to perform logical rule-based classification; we also seek this integration to be *tight* that these two normally-incompatible modules can be learned in an end-to-end manner, for which we propose the *BOAT* algorithm.

Compared with standard deep classifiers which use a linear model or variants of additive model as the classification head, Neural DNF provides a new choice of model based on logic rules. It offers interpretability via an explicit symbolic representation, strong model expressity, and a different type of model inductive bias. Neural DNF is particularly suited for certain tasks that require some logical composition and provides extra interpretability.

1 INTRODUCTION

In the recent years, the emphasis of machine learning, particularly deep learning, has been to increase the capacity to accurately model complex patterns, but typically such flexibility results in *blackboxes* that are too complex for humans to understand. While the ability of deep learning on feature extraction and pattern recognition is widely recognized, the blackbox nature makes it hard for human to interpret the underlying mechanism; furthermore, this lack of interpretability leads to difficulties for human to interact with or manipulate the model, when trying to debug undesired behaviors or to improve model by incorporating human knowledge.

A possible solution to this limitation of deep learning, as discussed and promoted by many researchers (Marcus, 2020; Bengio, 2019; Yi et al., 2019; Mao et al., 2019; Hudson and Manning, 2019; Penkov, 2019) , is to develop hybird neuro-symbolic models. While there are many different approaches on integrating neural and symbolic models, one of these approaches called **Vertical Integration**[1], aims at exploiting the best of both worlds in a straight-forward way: it utilizes deep learning to learn high-level features from raw sensory data (which deep learning are good at), and utilizes symbolic models to reason about the high-level features (which symbolic models are good at). More formally, the vertical integration approach can be represented 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 symbolic model processing the extracted features into final prediction. Vertical integration has its biological inspiration: we know that certain areas in the brain are used to process input signals (Grill-Spector and Malach, 2004) while others are responsible for logical reasoning (Shokri-Kojori et al., 2012). But from a more practical perspective, vertical integration promises a solution to demystify the blackbox by decomposing the prediction task into two stages, handled by different models; and a better-demystified *interpretable* model can enable easier human interactions and manipulations on the model.

[1]The term vertical integration is categorized in a recent survey (Garcez et al., 2019) on neuro-symbolic models. The neuro-symbolic literatute (Besold et al., 2017) offers a multilitude of approaches covering different settings, making it hard to discuss in brief due to space. We provide a more discussion in appendix.

The choice of symbolic model to integrate naturally depends on the task. SAT-Net (Wang et al., 2019a) utilizes a symbolic SAT solver layer on top of a convolutional neural network (CNN) for solving a satisfiability-based task of *visual Sudoku*; Donadello et al. (2017) utilizes first-order fuzzy logic on top of a Fast-RCNN to extract structured semantic descriptions from images; DeepProbLog (Manhaeve et al., 2018) builds a probabilistic logic program on top of a CNN within a domain-specified grammar template for tasks like visual digit addition and sorting. In this paper, since we deal with the most basic task of binary classification, we choose propositional logic in Disjunctive Normal Form (DNF), also known as 'decision rules' or 'rule set'. As a well-studied symbolic model, DNF has been established as being general and interpretable. DNF performs a simple and transparent 'OR-of-ANDs' prediction: if at least one AND clause (a conjunction of conditions) is satisfied, it predicts the positive class; otherwise negative. DNF is interpretable not only because its symbolic structure is intuitive to follow, but also that each conjunctive clause of DNF can be viewed as a separate IF-THEN rule, providing 'smaller-than-global' interpretations (Rudin, 2019). DNF is a general rule format, as any propositional logic formula has an equivalent DNF formula, and thus any rule-based binary classifier including decision set/list/tree can be expressed by a DNF.

We seek Neural DNF's vertical integration to be tight, i.e. we wish to optimize the deep feature extractor and the DNF end-to-end. However, the main technical challenge here is that the learning algorithms for rule-based models and deep learning models are generally incompatible, making end-to-end learning not directly possible. In order to address this challenge, we propose BOAT (Bi-Optimizer learning with Adaptively-Temperatured noise) to train ϕ and g jointly (see fig. 1): we use standard continuous-parameter optimizer Adam (Kingma and Ba, 2014) to optimize the first-stage neural network ϕ and modify a binary-parameter optimizer from Helwegen et al. (2019) to optimize the parameters of the second-stage rule-based 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. Compared with standard deep learning classifiers which has a fully-connected linear layer as g, Neural DNF offers (1) a logic-based inductive bias and (2) better interpretability at competitive accuracy.

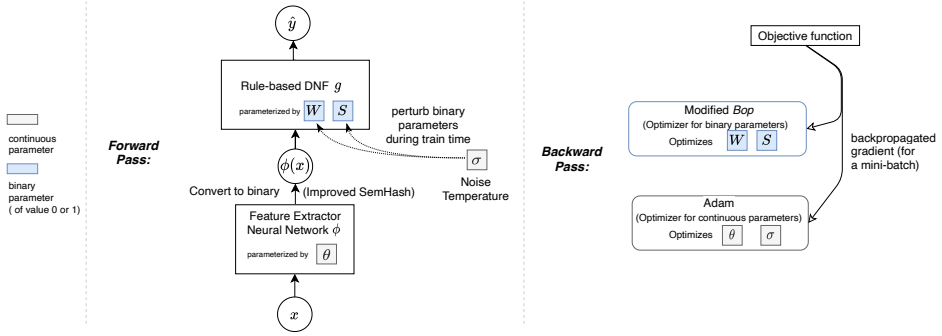

Figure 1: *Overview of Neural DNF and the proposed BOAT Algorithm*

The rule-based Neural DNF is particularly suited for classification tasks that require the logical compositions of certain concepts. We demonstrate that with experiments on a toy dataset called 2D-XOR, an extension of the XOR problem which has received special interest in the literature (Minsky and Papert, 1969). We consider 2D-XOR as a minimal example to show the benefits of Neural DNF's vertical integration, because 2D-XOR requires a model to both learn the right high-level features (concepts) from the raw data and the right XOR function. We further apply Neural DNF to image datasets in two scenarios: In the first scenario, we use a regular deep network as feature extractor with no further constraints. We show that Neural DNF can successfully learn both the feature extractor and the logical rules for classification, and achieve competitive accuracy. Note that in this case there is no guarantee that the extracted features are meaningful to human. In the second scenario, we constrain the feature extractor to produce human-aligned interpretable features by enforcing an auxiliary concept loss based on human concept annotations. In this scenario, Neural DNF becomes highly interpretable that the interpretability enables human to easily interact with and manipulate the learned model, such as performing human-intervention on the extracted features to improve accuracy, or slightly tweaking the model to recognize an imaginary class that does not exist in the dataset. In conclusion, our experiments show that Neural DNF achieves accuracy comparable that of blackbox deep learning models while offering an interpretable symbolic DNF representation, that makes human easier to interact with or manipulate the model.

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 later. We use the two-stage formulation : 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 Disjunctive Normal Form module g is used for the actual classification. The overall classification is given by $\hat{y} = f(x) = g(\phi(x))$.

The logical rule-based 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 candidate rules is constructed and then 'learning' corresponds to finding 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, we 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: 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 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, $\boldsymbol{W}_{2K\times N}$ is discarded and only the selected rules are stored. Conventional methods of rule learning (Lakkaraju et al., 2016; Wang et al., 2017) 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 is not compatible if we wish to jointly optimize g with the neural network ϕ end-to-end. In order to do so, we make two essential modifications: (I) we make both $\boldsymbol{W}_{2K\times N}$ and $\boldsymbol{S}_N$ learnable parameters; (II) we introduce a differentiable replacement[2] of the logical operation of eq. (1) so that gradients can be properly backpropagated:

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

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

eqs. (2) and (3) computes the DNF function exactly as eq. (1), but note that since $\boldsymbol{W}$ 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 $\boldsymbol{W}$ and $\boldsymbol{S}$ for each class (thus we have a different DNF 'tower' for each class) while the concept predicates being shared across classes; 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).

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. The neural network ϕ's output $\tilde{c}_i$ is real-valued. To ensure the extracted c can be processed by the DNF g, c needs to be binary. 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

[2]Obtaining eqs. (2) and (3) is not new, similar formulation can be found in the literature such as the logical activation functions (Payani and Fekri, 2019; Wang et al., 2019b) or soft NAND gate (Sajjadi et al., 2016).

training, we utilize the *Improved SemHash* (Kaiser and Bengio, 2018). It does not need any manual annealing of temperature (Bulat et al., 2019; Hansen et al., 2019) or additional loss functions and has been demonstrated to be a robust discretization technique in various domains (Kaiser and Bengio, 2018; Chen et al., 2019b; Kaiser et al., 2019). Ideally, the intermediate output $\boldsymbol{c}$ should be the high-level 'natural' features that are aligned with human-comprehensible concepts. Quoted from Melis and Jaakkola (2018), 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 indeed a very much more challenging task. We explore two options of ϕ on some image datasets: in section 4.2. we choose to use a convolutional neural network and do not put any extra constraints; in section 4.3, we enforce the feature extractor to produce interpretable representations by enforcing an auxiliary concept loss based on human annotations.

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 the classification loss (using MSE or BCE) and $\mathbb{R}_g$ is the regularization for g. For a simple regularization of g, we want the number of rules and the length of rules to be small. Note that as only rules selected by $\boldsymbol{S}$ are actually used, we can use a grouped L1-norm by $\mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S}) = \lambda_g \sum_j^N |\boldsymbol{S}_j|_1 |\boldsymbol{W}_{\cdot,j}|_1$. Note that in the multi-class setting, our extension of Neural DNF is simply treating a multi-class classification as multiple one-versus-all binary classifications. In this case, the objective function sums up the classification loss for all the classes.

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 utilizes two optimizers: a standard deep learning optimizer Adam (Kingma and Ba, 2014) that optimizes the continuous parameters θ of the neural network ϕ and a binary-parameter optimizer adopted from [Helwegen et al., 2019] that optimizes the binary parameters $\{\boldsymbol{W}, \boldsymbol{S}\}$ of the DNF g. As the key ingredient of BOAT, during training 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 appendix fig 1) and (2) the noise temperature σ is also a learnable continuous parameter, which can be optimized together with other continuous parameters by Adam. Making the temperature learnable avoids the tedious tuning of temperature schedules.

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 (Payani and Fekri, 2019; Wang et al., 2019b), 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 (Courbariaux et al., 2016; Darabi et al., 2018). 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) (Bengio et al., 2013). 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.

The introduced BOAT combines the best of both approaches: (1) It is optimization-friendly and not very sensitive to initialization and hyperparameters. This is especially important as g is not overparameterized; (2) It naturally optimizes binary parameters. A pseudocode is provided in appendix.

Modified Bop: BOAT uses a modified version of the *Bop* optimizer (Helwegen et al., 2019) to optimize the binary-valued parameter $\{\boldsymbol{W}, \boldsymbol{S}\}$ given gradient as learning signals. We make minor modifications to suit *Bop* (originally for $\{-1, 1\}$) 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 mini-batch gradient. We use $\gamma = 1 - 10^{-5}$ and $\tau = 10^{-6}$ as default value while the trick for tuning γ and τ can be found in original *Bop* paper.

Key ingredient of BOAT: 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 the introduced noise smoothes the loss surface and helps the optimization. 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:

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

During training the perturbed weight $\tilde{w}$ is used in the forward pass computation of the objective function; in test time, we simply disable this perturbation. 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 make σ_w also a learnable parameter. We initialize $\sigma_w = \sigma_0$ and optimize σ_w by Adam as well. The choice of initial value σ_0 requires heuristic: with a too large σ_0 optimization becomes slower (appendix fig 1c), and σ cannot be too small: in the extreme case with zero noise the optimization of $\boldsymbol{W}, \boldsymbol{S}$ will constantly fail (appendix fig 1b). We find values in $[0.1, 0.3]$ all work fine and we use $\sigma_0 = 0.2$ as the default initial value. **Remark**: We evaluate BOAT with its alternatives for learning the DNF alone in appendix sec E.1. We can also apply same noise for DNF-STE, but it converges slower (appendix fig 2). 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 main advantage of *Bop*.

4 Experiments

4.1 On a 2D toy dataset.

Here we first apply Neural DNF on a 2D toy dataset 2D-XOR (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. We consider 2D-XOR as an extension of the historically important XOR problem, and its difficulty is that the feature of being 'red'/'non-red' and 'square'/'non-square' is not provided as input directly but needs to be learned in a 2-d input space. This makes the learning harder since it requires to learn both the correct intermediate feature and the correct logical decision function. We use a one fully-connected layer network as ϕ to produce two concept predicates c_1, c_2 and can visualize in as two lines in the 2-d space. 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 (1) the feature used for logic-based function are not provided but needs to be learned and (2) the underlying classification process consists of subtypes (each described as one rule) and requires some logical compositions.

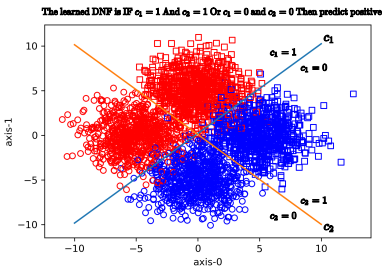

Figure 2: Neural DNF on 2D-XOR

4.2 ON IMAGE DATASETS. SCENARIO 1

In this section we apply Neural DNF with a CNN as feature extractor without any further contraints. and then demonstrate empirically that given the same first-stage feature extractor architecture, Neural DNF sacrifices slight loss of accuracy (table 1) while gaining more faithful interpretations (fig. 3). We first evaluate the accuracy of Neural DNF on several image datasets table 1. Using the same architecture for ϕ, the models we compare are (1) Neural DNF (2) a standard DNN (i.e. a linear model as g) and (3) self-explaining neural network (SENN) (Melis and Jaakkola, 2018), which uses a neural network to generate the coefficients of a linear model conditioned on the input and claims to have better interpretability. Since binarization is not required for DNN or SENN, we also evaluate DNN and SENN with/without it. We observe that all models perform similarly well in terms of test accuracy across datasets, while Neural DNF loses accuracy only slightly. Comparing DNN and SENN with/without binarization, we can see that both DNN and SENN loses accuracy slightly because of the binarization. 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. We suspect this is also partially because Neural DNF treats multi-class settomg as multiple one-versus-all classifications and does not produce probabilistic outputs like DNN and SENN. However, this should not be viewed as a weakness as long as the accuracy loss is slight. We believe that this issue can be alleviated that given a sufficiently flexible ϕ, the accuracy loss of Neural DNF can be negligible like the case of simple dataset MNIST.

Table 1: Test Accuracy on some image datasets

	MNIST	KMNIST	SVHN	CIFAR10
Neural DNF	99.08%	95.43%	90.13%	67.91%
standard DNN	99.12%	95.86%	90.74%	70.43%
standard 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%

Now regarding the interpretability, the problem here is, we can in principle inspect[3] the meaning of concept and rules of Neural DNF, but since we have no constraints on ϕ, the extracted features are only highly discriminative and is not guaranteed to be aligned with human perceptible concepts. A symbolic DNF that operates on non-interpretable representations is still non-interpretable. We will revisit this interpretable representation problem in next section; but despite tje first-stage, here we can still compare the interpretability of the second-stage model. Despite many qualitive arguments for favouring the interpretability of the symbolic DNF (appendix B.3), we can further quantitively evaluate the faithfulness of models' interpretability, a critical critera that measures how faithful the explanation for a particular prediction are to the underlying computation of prediction. We adopt the *faithfulness metric* proposed by (Melis and Jaakkola, 2018) 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.[4] We report the faithfulness metric on test set for DNN, SENN and Neural DNF in fig. 3. For DNN, we use the coefficients of final linear layer as rele-

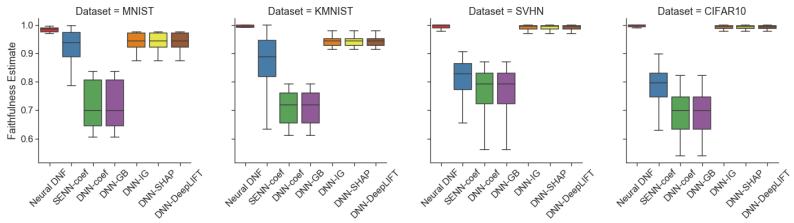

Figure 3: Evaluating the faithfulness of explanations on test set

vance scores. Alternatively, for DNN we can also utilize some representitive post-hoc interpretation methods (also known as attribution methods): Guided Backprop (GB) (Springenberg et al., 2014),

[3]We do provide an anecdotal example on the explanations Neural DNF can derive in (appendix fig 4) as a direct comparison to the explanations provided by linear models (e.g., the MNIST example from the self-explaining network (Melis and Jaakkola, 2018)) and explains the benefits of a symbolic DNF.

[4]As in (Melis and Jaakkola, 2018), the relevance score assignment and feature perturbation is done for the extracted feature $\phi(x)$, not at the level of the raw data x.

gradient SHAP (Lundberg and Lee, 2017), Integrated Gradient (IG) (Sundararajan et al., 2017) and DeepLIFT (Shrikumar et al., 2017). SENN can use its self-generated coefficients as relevance scores (Melis and Jaakkola, 2018). For Neural DNF, we use the difference analysis (Robnik-Šikonja and Kononenko, 2008), a model-agnostic relevance score assignment method to assign scores. As shown in fig. 3, 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 highly faithful in a robust way across different test samples. SENN is more faithful than DNN; and for the post-hoc methods while GB performs very similarly with DNN-coef, IG/SHAP/DeepLIFT drastically improves the faithfulness. This is because the latter three methods compute the relevance w.r.t to a baseline reference, a concept that is now considered very essential (Sturmfels et al., 2020). However, we emphasize 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 symbolic nature of DNF.

4.3 On image datasets, scenario 2

Here we test a new scenario where we have concept annotations so we can train the feature extractor to produce human-aligned interpretable representations. Our goal is that if we can align the extracted features with human understanding, then we can achieve a highly interpretable model which human can easily interact with and manipulate. We use the CUB dataset (Wah et al., 2011) which has 200 class and 112 binary annotataed concepts (preprocessed by Koh et al. (2020)). In Koh et al. (2020) the authors propose the concept bottleneck model, a similar two-stage model using a Inception-V3 based feature extractor ϕ and a linear layer g. Given concepts the annotations, a concept prediction loss can be applied so that the extracted faetures are constrained to align with annotated values. Koh et al. (2020) also propose the *test-time human intervention*: since the extracted concepts can be wrong, human can check and correct extracted concepts so that test accuracy can be improved significantly through this interaction. We follow Koh et al. (2020) and replace the second-stage g with a DNF. Koh et al. (2020) evaluate several strategies for training the two-stage model. Here we use the independent training strategy: we train ϕ to predict concepts and train g to predict class label using concepts; only in test time we stack ϕ and g together. The reason of choosing independent training instead of joint training is counterintuive, however, this is because we find that the extracted concepts after joint training are less well-aligned with human annotations, and it then makes the human intervention less effective than independent trained models. In other words, based on non-interpretable features, any human interaction/manipulation will be unreliable. This phenomenon is consistent with Koh et al. (2020), and we think novel architectures that extracts interpretable features without the need of annotations can solve it, in that case joint training should give better results. In table 2 we report the accuracy of Neural DNF, the concept bottleneck model and blackbox DNN. Neural DNF achieves less test accuracy compared with the concept bottleneck model by a large margin, we suspect it might because Neural DNF is dealing with too many (200-class) one-versus-all prediction and does not produce probabilistic outputs like other models do. But after intervention both Neural DNF and the concept bottleneck model achiveve perfect accuracy. This indicates that (1) the accuracy by applying human intervention can be improved significantly which is the benefits of having an highly interpretable model; (2) the bottleneck of accuracy is the feature extractor: the classification of CUB can be effectively solved given a perfect feature extractor.

Table 2: Test Accuracy on CUB dataset

	test acc	test acc with human intervention
Neural DNF	61.94%	100.00%
Concept bottleneck model	72.38%	100.00%
Blackbox DNN	74.69%	N/A

While losing quite much in accuracy, the symbolic nature enables the Neural DNF to do the things the concept bottleneck model cannot. We show in fig. 4 (top) a correctly predicted sample for the class of 'Chestnut sided Warbler' and in fig. 4 (middle) an incorretly predicted sample and how human intervention can correct the prediction. DNF rules are a very human-readable format, that the rules are very sparse involving only a few concepts and the logical operation is intuitive to understand. The DNF rules also alleviate the burden of human intervention compared to the concept bottleneck, as only a few concepts are used in the decision rule, a human user can only check and correct these concepts indicated in the rules and does not have to go through every concepts which

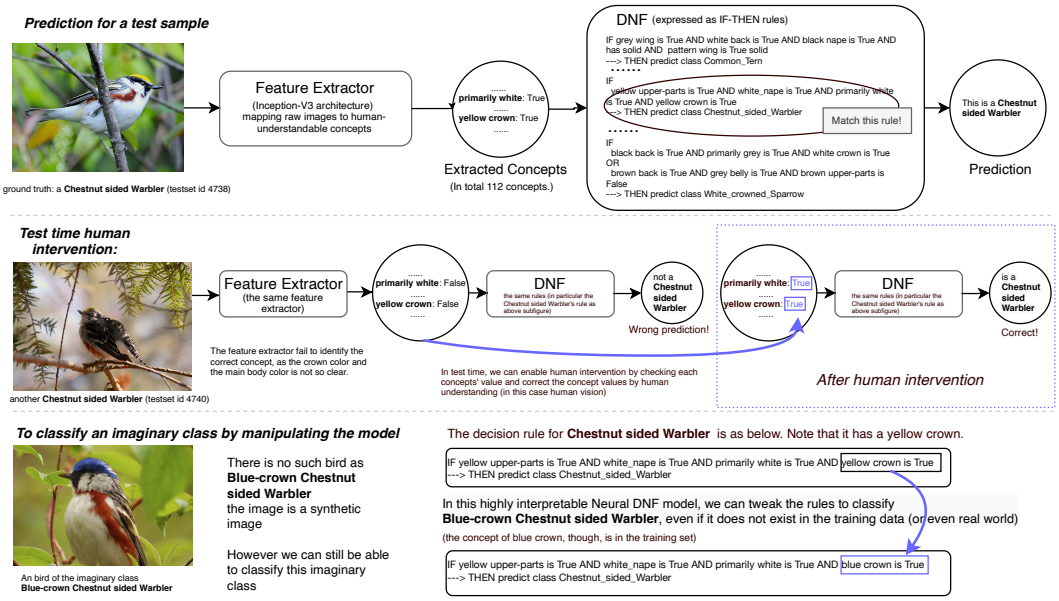

Figure 4: (top) illustration of a correct prediction of Neural DNF for class 'Chestnut sided Warbler'; (middle) illustration of human interaction: an incorrect prediction of Neural DNF for class 'Chestnut sided Warbler' can be corrected by human intervention; (bottom) illustration of manipulating a learned Neural DNF to classify an imaginary class 'Blue-crown Chestnut sided Warbler'.

the concept bottleneck model requires. Also, the symbolic nature of DNF enables us to incorporate knowledge and manipulate the models easily in a way that the concept bottleneck model (with a linear g) cannot offer. In in fig. 4 (bottom), we provide an illustration on how we can tweak the Neural DNF to predict an imaginary class "blue-crown Chestnut sided Warbler" that does not exist in the training dataset (not even in real world). Taking the rule for 'Chestnut sided Warbler', we can simply replacing the condition of 'yellow crown is True' with 'blue crown is True'. Note that 'blue crown' is already in the training set and one of the concepts ϕ can extract. Of course we can go further that we can train and append new feature extractors for new concepts and then play with new concepts. There are, however, limitations because of the expressity of prositional logic.

5 Conclusion and Future Work

In this paper, we have presented Neural DNF, a vertically integrated neuro-symbolic classifier. Neural DNF takes a step towards the fundamental problem of integrating continuous perception and logical reasoning. Neural DNF adopts a two-stage model that utlizes a DNN to extract features, called concept predicates, and a propositional logic DNF module to make classification based on the concepts. We propose the BOAT algorithm for joint learning of the DNF module and the feature extractor DNN. In order to avoid laborious manual tuning, we employ an Improved SemHash method to binarize the extracted features, to obtain concept predicates, because it does not need to manually tune any extra parameters or add further losses. We also introduce adaptive temperatured-noise, which is a minimal modification of the binary-parameter optimizer *Bop* that enables the effective optimization of the parameters of the second-stage rule-based model.

This paper suggests several directions for future research. In our experiments in Neural DNF , we have used concept annotations and the concept-bottleneck DNN architecture to extract interpretable features. However, in practice there is often no prior knowledge of 'concept annotations' available. Therefore, novel deep learning architectures that can extract human-understandable concepts from data without the need of concept annotations should be investigated. This may require domain-specific knowledge representations and loss functions (Melis and Jaakkola, 2018; Chen et al., 2019a; Biffi et al., 2018; Kim and Canny, 2017; Johnson et al., 2016). Another future direction is the use of more powerful languages than propositional logic, for example, the more general inductive logic programming. 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.

REFERENCES

Yoshua Bengio. From system 1 deep learning to system 2 deep learning. *Neural Information Processing Systems. December 11th*, 2019.

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

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.

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.

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.

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, 2019a.

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, 2019b.

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.

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

Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

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.

Kalanit Grill-Spector and Rafael Malach. The human visual cortex. *Annu. Rev. Neurosci.*, 27: 649–677, 2004.

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.

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.

Drew Hudson and Christopher D Manning. Learning by abstraction: The neural state machine. In *Advances in Neural Information Processing Systems*, pages 5903–5916, 2019.

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.

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

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.

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.

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

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. *arXiv preprint arXiv:2007.04612*, 2020.

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.

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.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018. URL `http://arxiv.org/abs/1805.10872`.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

Gary Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

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.

M Minsky and S Papert. Perceptrons: An introduction to computational geometry. 1969.

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

Svetlin Valentinov Penkov. *Learning structured task related abstractions*. PhD thesis, University of Edinburgh, 2019.

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

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.

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

Ehsan Shokri-Kojori, Michael A Motes, Bart Rypma, and Daniel C Krawczyk. The network architecture of cortical processing in visuo-spatial reasoning. *Scientific reports*, 2:411, 2012.

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.

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

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

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.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

Po-Wei Wang, Priya L Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019a.

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.

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

Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. *arXiv preprint arXiv:1910.01442*, 2019.