Supplementary Material

Anonymous Author(s)
Affiliation
Address
`email`

The appendix mainly consists of 5 parts.

- Appendix A: We present details about the key ideas and techniques used in the paper but is not discussed in details due to page limitsm, including details of the model Neural DNF and the learning algorithm BOAT.
- Appendix B: We discuss related works on interpretability research. we first answer why we prefer inherently interpretable models, we only mention briefly in the main paper, here we discuss more. We then talk about recent works on interpretable deep learning. We end with a discussion of a comparision of interpretability of rule-based models and linear model.
- Appendix C: We discuss more about Disjunctive Normal Form (DNF). for classification, previous works on learning DNF and more closely, some recent works to learn DNF by gradient descent using a differentiable DNF function.
- Appendix D: We relatin Neural DNF to the literature of neuro-symbolic integration.
- Appendix E: We present details of experiment setup and experimental results not covered in the main paper.

Contents

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

A Algorithm Details

we introduce some key ideas and techniques and the details of algorithm

A.1 The modified *Bop*

Helwegen et al. [2019] proposes the *Bop* optimizer in the context of binarized neural networks of value $\{-1, 1\}$. As suggested by Helwegen et al. [2019], the Bop can be viewed as a basic (gradient-based) binary optimizer, in the same sense that SGD is a basic (gradient-based) continuous-valued optimizer. The update rule for the original Bop is

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

And the modified bop for $\{0, 1\}$ used in this paper is

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

The modification of suiting the case of $\{-1, 1\}$ to $\{0, 1\}$ is minor.

Also, in the implementation we add a bias correction procedure. Let $\hat{m}_t$ be the non-corrected gradient momentum that is updated as $\hat{m}_t = \gamma \hat{m}_{t-1} + (1 - \gamma)\nabla$, The bias correction is given by

$$m_t = \hat{m}_t / (1 - \gamma^t)$$

We need this correction because in the original Bop paper Helwegen et al. [2019] uses random initialization for m but in our implementation we make specific that m is zero-initialized. So we will need this bias correction.

A.2 BOAT

Basically, BOAT consists of the modified bop and the proposed temperatured noise. We differ from the Helwegen et al. [2019] as follows: (1) First, we fit into the case of $\{0, 1\}$ instead of $\{-1, 1\}$, which is trivial; (2) we introduces the adaptively-temperatured noise controlled by the learnable temperature parameter. (3) we add a bias-correction procedure.

We mentioned in the main paper that we suspect the noise smoothes the loss surfaces so to help the optimization. But it is far from a rigorous statement and we do lack some theoretical understanding of why and how this noise helps learning.

We refer interested readers to the original Bop paper Helwegen et al. [2019] or a more recent paper by Meng et al. [2020] who discusses the connection of Bop and STE-Binary network and also provides a bayesian perspective on binary network learning. We believe that the our method BOAT can in principle be linked to approximate variational inference [Khan et al., 2018, Kingma et al., 2015, Potapczynski et al., 2019]. In particular, the variational Adam [Khan et al., 2018] is very like ours, except they works for continuous values and we add noise for binary parameters. We leave further theoretical investigation of BOAT for future works.

A.3 Improved SemHash

Since the 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:

$$c_i = \begin{cases} 1, & \text{if } \tilde{c_i} > 0 \\ 0, & \text{otherwise.} \end{cases}$$

However, since gradient through this step function is zero almost anywhere and thus prevents training, we utilize the *Improved SemHash*[Kaiser and Bengio, 2018] as one way to make the overall model differentiable.

During training, Improved SemHash first draw Gaussian noise ϵ with mean 0 and standard deviation 1. The noise ϵ is added to $\tilde{c}$, two vectors c and c' are then computed.

$$c = \mathbf{1}(\tilde{c} + \epsilon)$$

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 ($\{\boldsymbol{W}, \boldsymbol{S}\}$); The neural network ϕ (θ).

```
 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 $\sigma_w$ (??).
 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 $\sigma_w = min(0.5, max(\sigma_w, 0))$
```

c is the result after applying the binary step function and

$$c' = max(0, min(1, 1.2 * sigmoid(\tilde{c} + \epsilon) - 0.1))$$

c' is computed by the above function called saturating sigmoid function [Kaiser and Sutskever, 2015, Kaiser and Bengio, 2016]. During training, c is used half of the time and and c' is used for the other half of the time in the forward pass; for the backward pass we define the gradient of c to $\tilde{c}$ the same as c' to $\tilde{c}$. During testing, the noise is disabled and c is used as output.

It is not clear according to the description of [Kaiser and Bengio, 2018] what 'half of time' for c and c' means, that is, whether we use c for one mini-batch and c' for the next mini-batch, or we use c and c' for half of the samples for each mini-batch. In our implementation, we choose the latter option: we use c for half of the samples in the mini-batch, and use c' for the other half of samples in the mini-batch, determined randomly.

The use of saturating sigmoid function, instead of just sigmoid function, is introduced first by Kaiser and Sutskever [2015] who claims to have slight improvement. But we do not observe such improvement in Neural DNF so in our implementation we simply computed

$$c' = sigmoid(\tilde{c} + \epsilon)$$

We use the simple sigmoid function instead of the saturating sigmoid function.

We name two reason of using Improved SemHash: (1) Improved SemHash does not need any manual annealing of temperature [Bulat et al., 2019, Hansen et al., 2019] or additional loss functions. There are some serveral alternative that need tuning of annealing, including the annealed sigmoid/tanh [Bulat et al., 2019] and gumbel-softmax trick [Maddison et al., 2016], but we believe tuning this annealing schedule is difficult. (2) Improved SemHash achieves good results compared with many alternative solutions. Comparisons with other discretized latent representation learning can be found at [Kaiser et al., 2018], semHash performs great despite being very simple. It also has been demonstrated to be a robust discretization technique in various domains [Kaiser and Bengio, 2018, Chen et al., 2019b, Kaiser et al., 2019]. But of course, we note that Improved SemHash is not the only option for binarizing the extracted features.

A.4 Initialization of $\tilde{W}$

We will initializes $\boldsymbol{W}$ and $\boldsymbol{S}$ randomly, each entry is drawn from a Bernoulli disctribution where $p_{\text{Bernoulli}} < 0.1$. This is because we want $\boldsymbol{W}$ and $\boldsymbol{S}$ to be sparse.

Remark: note that if we are really careful about the initialization, there is a small issue for initialization of $\boldsymbol{W}$, because some w can the value of c or the negation value. In principle, a condition and the negation of it cannot be set to 1 at the same time; but we do not consider it in our implementation. It seems that it does not matter in our optimization using BOAT .

A.5 Regularization of DNF: R_g and possible Alternatives

The simplest choice of $\mathbb{R}_g(W)$ is to use a L2 regularization. However, this is not what we really want. Recall that we wish to obtain a DNF, a set of rules where the number of total rules is small and the length of each rule is small. So more technically we want a small number of columns of $\boldsymbol{W}$ that have non-zero elements and small number of non-zero elements for each column (that is indicated by the membership vector $\boldsymbol{S}$). This can be realized by penalizing the number of rules as the L1-norm of $\boldsymbol{S}$ and the length of selected rules also by the L1-norm (note that as only rules selected by $\boldsymbol{S}$ are actually used), we use a grouped L1-norm by

$$\mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S}) = \lambda_g \sum_{j}^{N} |\boldsymbol{S}_j|_1 |\boldsymbol{W}_{\cdot,j}|_1$$

which is very like group lasso.

The reason we use this the grouped L1-norm is that does variable selection at the group level and in our case can effectively eliminate a group of weights by columns(rules). We support the use of grouped L1-norm (like group lasso) instead of L2 norm by an empirical study on the cognitive preference of rules Fürnkranz et al. which shows that there is no strong preference on shorter rules but instead even a slight preference on longer rules. In other words, a small number of long rules is preferred over many short rules.

We leave more sophisticated metrics like feature overlaps [Lakkaraju et al., 2016] as future work. There are, indeed, many metrics but it remains hard to apply them for Neural DNF because some of them is not differentiable. If we use a discrete optimization algorithm this is not a problem, but in the case of our Neural DNF , we need the objective function to be end-to-end differentiable, so we need the regularization terms to be differentiable as well.

A.6 Possible direction of improvement

In the main paper we only discuss the regularization for the second stage DNF g, so the overall objective is given as

$$\boldsymbol{L} = \mathbb{L}_{loss} + \lambda_g \mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S})$$

This is because we do not focus or design anything for the first stage neural network feature extractor. If we consider this (for future works), we can extend the objective to

$$\boldsymbol{L} = \mathbb{L}_{loss} + \lambda_g \mathbb{R}_g(\boldsymbol{W}, \boldsymbol{S}) + \lambda_\phi \mathbb{R}_\phi(\theta)$$

by considering a regularization term for the feature extractor that somehow defines some interpretability constraints.

B Related Works on Interpretability

The interest for interpretability is not new. It appears with the development of rule-based expert systems in the mid-1980's Amel [2019]. Of course the current situation is different because recently we have seen an increasing trend of interpretability research in machine learning [Lipton, 2018], in particular interpretable deep learning[Xie et al., 2020, Fan et al., 2020].

There are many definitions on interpretability, we use Lipton [2018]'s Simulatability definition of interpretability: for a prediction to be fully understood, the human should be able to re-calculate and reach the same prediction given resonable time.

B.1 Inherent interpretability, not Post-hoc interpretation

We summarize two main tracks of approaches for interpretability research, namely *post-hoc interpretation* and *building inherently interpretable models*, and we favor the latter. Post-hoc interpretation builds a secondary model to explain the given pre-trained deep learning model. Representative works include saliency maps [Simonyan et al., 2013], LIME [Ribeiro et al., 2016], Concept Activation Vectors [Kim et al., 2017]; some aim at giving counterfactual explanation [Dhurandhar et al., 2018, Zhang et al., 2018, Grath et al., 2018]. However, the provided explanation is in fact provided by a secondary model, not the original one, so it might not correspond faithfully to how the original blackbox model actually computes its prediction. Recent works further suggest that post-hoc interpretations are not robust [Adebayo et al., 2018, Fen et al., 2019, Alvarez-Melis and Jaakkola, 2018] and can even be misleading [Lakkaraju and Bastani, 2019, Rudin, 2019], and more specifically, the counterfactual explanations have the 'unjustify' issue [Laugel et al., 2019].

The second track, on the contrary, is to build an inherently interpretable deep learning model, so that the explanation it provided is exactly how it calculates the prediction. Inherently interpretale models are more preferred to examine and use when deployed in real world applications for various advantages (See [Khandani et al., 2010, Florez-Lopez and Ramon-Jeronimo, 2015]): it derives explanations to justify decision (legal issue), so people are less likely to refuse to use, easier to be combined with experts. An inherently interpretable classifier uses a interpretable prediction process to compute the prediction, and provides such computation process itself as the explanation for prediction.

B.2 Related works on Interpretable Deep Learning

For tabular datasets where each feature is already-meaningful, linear model or rule-based model are well-established choices of interpretable models[Amel, 2019], but for other data types such as image where each feature dimension itself is not meaningful, interpretable model is harder to construct.

A reasonable approach, which we call the two-stage paradigm, is to first construct intermediate representations $\phi(x)$ that is interpretable, and on top of that a simple interpretable classifier g is applied as the second stage such that the prediction is given by $\hat{y} = f(x) = g(\phi(x))$.

Most works on interpretable deep learning [Melis and Jaakkola, 2018, Chen et al., 2019a, Vaughan et al., 2018] choose linear model as the second-stage model g. Indeed, any neural network can be intuively viewed as a two-stage model where the second-stage is a linear model, as long we treat the network up to the last hidden layer as a generic feature extractor ϕ. But the notion of interpretability means we wish to make certain heuristics to make $\phi(x)$ interpretable.

Vaughan et al. [2018] formularize g as a linear model that $g(\phi(x)) = \sum_i w_i \phi_i(x)$ where $\phi(x)$ is a set of ridge functions, each of which are produced by a neural network. It claims to be more interpretable than general networks, because such a function has simpler partial derivatives that can simplify saliency analysis, statistical analysis and so on.

Chen et al. [2019a] also uses a linear model g and proposes a prototype-based design for $\phi(x)$. It learns interpretable $\phi(x)$ in the sense that each dimension of $\phi(x)$ is the similarity score to an image patch. It provides extra interpretability tailed for vision tasks as the similarity scores $\phi(x)$ can be visualized with the corresponding 'prototype' image patches. Melis and Jaakkola [2018] takes a step further that they use neural networks to produce not only $\phi(x)$ but also the coefficient of the linear model, such that prediction function is given by $g(\phi(x)) = \sum_i w_i(x)\phi_i(x)$ where $\phi(x)$ (called 'concepts') are regularized by auto-encoding reconstruction loss and $w_i(x)$ can be intuitively understood as an input-dependent relevance score for a concept $\phi_i(x)$. Here we can see that the implementation of $\phi(x)$ is domain-specific and customizable.

Theses works extend standard deep neural networks to interpretable ones by improving the design of the feature extractor ϕ and use a linear model for g. This is because a linear model, or its general form of general additive model, can be integrated into the automatic differentiation very easily.

Taking an opposite direction, we differ with previous works by considering to a rule-based classifier for g and do not put our focus on ϕ. However, because of the discrete struture, integrating a rule-based model g remains an hard and unexplored direction. But in terms of interpretability we favor rule-based models over linear models, not only because the discrete structure is more intutive for human to follow, but also that it can provide only the satisfied rules for explanation, unlike that for linear model for which we need to present the full model coefficients.

B.3 why favouring the symbolic DNF for interpretability

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 [Freitas, 2014, Huysmans et al., 2011, Wachter et al., 2018] 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 [Doshi-Velez et al., 2017, Wachter et al., 2018, Grath et al., 2018, Miller, 2019].

On the other hand, coefficients as feature importances are arguably 'harder to use and to understand for a non-expert user' [Wachter et al., 2018]. 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 [Bruner et al., 1956]. (2) **Using rules as a second-stage 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 future works

Rules can derive counterfactual explanations while linear models cannot (at least, not easy for linear models). The explicit decision boundary of rules not only can give factual explanations, namely giving the conjunctive conditions of the satisfied rule, but also give counterfactual explanations, by simply presenting that could have changed the prediction. Unlike weighted feature importance, rules explicitly presents all the sufficient conditions for prediction and thus can naturally handles counterfactual explanations in the form of *'Would changing a certain factor have changed the decision?'*. Indeed, the counterfactual explanation is often considered to be the most important property of explanation, confirmed not only from a more practical perspective [Doshi-Velez et al., 2017, Wachter et al., 2018, Grath et al., 2018] and also from cognitive/psychological research [Miller, 2019].

C DNF

Here we first explain the reason of choosing DNF as the rule module.

We choose DNF, one of the most powerful and historically significant symbolic methods, for its interpretability and its generality. It has a simple and transparent '*OR-of-ANDs*' prediction process: if at least one *AND* clause (a conjunction of Boolean predicates) is satisfied, it predicts positive; otherwise it predicts negavtive. DNF is interpretable not only that the discrete structure is intuitive to follow, each conjunctive clauses (AND) of DNF can be viewed as subtype for explanation, i.e. the DNF can be decomposed into individual local patterns. We also appreciate the generality of DNF, as any propositional logic formula has a equivalent DNF and thus any rule-based binary classifier including decison set/list/tree can be expressed as a DNF.

DNF have a long history but learning DNF is still a very hard problem, not to mention that we need to add interpretability contraints. Theoretical results on learning DNF in the PAC (probably approximately correct) setting are often unrealistic in practice and are hard to incorporate extra interpretability contraints. On the other hand, the practical use of rule learning of a DNF form attracts more attention from data mining community, namely, desciptive pattern discovery [Novak et al., 2009]. Seminal algorithms include CN2 and RIPPER (constrained for binary classification). More recent state of the art machine learning algorthms for learning DNF can work quite well on small tabular datasets, but not very scalable on high dimensional datasets, we name few representative recent works using greedy heuristics [Obermann and Waack, 2015, 2016]. Bayesian approximation [Wang et al., 2015], linear programming [Su et al., 2015]. We name the interpretable decision set by Lakkaraju et al. [2016] and Bayesian rule set by Wang et al. [2017] as two recent work as representative.

However, we note that these approach is not compatible here if we wish to jointly optimize g with the neural network feature extractor ϕ in an end-to-end way. It is because that first, constructing $\boldsymbol{W}$ by rule mining becomes non-sense if the neural module is currently being trained; second, discrete optimization methods for learning $\boldsymbol{S}$ is not compatible with gradient-based optimization.

C.1 Differentiable replacement of the DNF function g

The differentiable replacement of the DNF function we use in the paper is adopted from Payani and Fekri [2019], Wang et al. [2019b]. However, as we mentioned in the footnote in the main paper, we believe that this relaxation is not new but rather re-invented. Simialr formulations on differentiable operations see [Sajjadi et al., 2016, Arabshahi et al., Nomura et al., 1992]. It is also likely that we miss critical references on neuro-fuzzy system research in the 90s.

Basically, we construct differentiable g following the approaches of [Payani and Fekri, 2019, Wang et al., 2019b], where neural networks that execute differentiable logical operations are learned for inductive logic programming [Payani and Fekri, 2019] and multiple DNF layers are stacked to classify on tabular datasets [Wang et al., 2019b].

We here introduce some works that also uses a similar differentiable replacement of the DNF function below. Sajjadi et al. [2016] learns a DNF in the context of tackling the *moving target* problem (herd-effect) [Fahlman and Lebiere, 1990]. For each rule there is no selection of conditions, but rather, all conditions are taking as conjection. One thing worth mentioning is that Sajjadi et al. [2016] do not actually train the parameters of the disjunctive function but only gives a good initialization.

Payani and Fekri [2019] propose the *neural logical networks* to solve inductive logic programming, optimized in a gradient-based way. It does not add any extra regularizations to push the parameters to binary values, in the other hand it carefully discusses the problem of initialization to solve the problem.

Wang et al. [2019b] proposes to stack many DNF layers to make accurate and interpretable predictions (at least the authors claim that the resulted networks is interpretable), although we doubt it as the stacked DNF may in fact corresponding to a rule set that is so complicated for human too understand. Wang et al. [2019b] also propose the randomized binarzation to make the parameters of the matrix $\hat{\boldsymbol{W}}$ to be close to binary values. We view this as a binarization version of dropout, at each round, some parameters in $\hat{\boldsymbol{W}}$ are thresholded to 0 or 1 and get fixed, and other parameters are optimized as usual by backprop. However, it seems in the experiment of [Wang et al., 2019b] that the rate of binarization plays an important role, often the rate need to set in a very high value (for example, 80 90%) We also think that, it is not very efficient in terms of learning, by fixing such a high percentage of parameters to thresholded values. Because there binarized parameters are not getting updated at all (no gradient).

Optimization issues. The presense of both discrete and continuous parameters in Neural DNF introduces a hard optimization problem. A straightforward solution is to use a EM-like approach, by applying well-studied optimization techniques for neural and rule-based module separately. But we argue such separate training scheme might not be as efficient as joint end-to-end training. Another solution is to end-to-endly optimizing the overall model, which is challenging. We discuss in the main paper of two straightforward alternatives for joint learning: DNF-Real and DNF-STE and we also propose BOAT , the key algorithm in our paper.

Possible future directions. We consider possible directions about moving from DNF (propositional logic) to more flexible and powerful rule languages, And we are hoping that the proposed BOAT can possibly help.

For example, there is an inductive program synthesis project called 'TerpreT' [Gaunt et al., 2016], where the authors design a special language TerpreT for inductive program synthesis. It is designed to separate the program specificaion and the inference algorithm so that it can be optimized by different backends such as SGD, relaxed linear programming and non-machine-learning solvers so that compare all these optimization backends. Its key surprising finding is that in terms of empirical performance, SGD is dominated by more traditional constraint solvers. It also gives a very simple case where SGD can fail consistently with a exponential number of local minima, yet constraint solvers can work it out very easily. Note that in [Gaunt et al., 2016], the binary parameters is not in fact binary but only transformed by sigmoid/tanh/etc, so it is like the DNF-Real [Payani and Fekri, 2019] we mentioned in the main paper.

D Connections to Neural-symbolic Integration

It turns out that interpretable deep learning is very close to neuro-symbolic integration, just slightly different focus. Neural DNF can also be seen as achieving an interpretable model by neuro-symbolic integration. In fact, this 'explainability or interpretability through neuro-symbolic integration' approach is recently gaining attention (for example, see David Cox's AAAI/IAAI 2020 invited talk).

Neuro-symbolic integration [Besold et al., 2017], which aims at combining neural methods with symbolic-logic methods, learning and reasoning, seems to be very related to our Neural DNF . However, the literature on this topic is vast and offers a multitude of approaches covering different settings, making it difficult to discuss related works completely. We follow a most recent survey [Garcez et al., 2019], which divides neuro-symbolic integarion into the two categories: Horizontal and Vertical.

Horizontal integration contains most of the research, aims at integrating neural and symbolic techniques into one inseparable model: either by making neural model behaves more like a symbolic model (also known as deep learning with logical contraints), which using logical knowledge to improve neural network learning, or the making symbolic method neural, using a neural network as interpreter to execute symbolic programs. Some approaches for Horizontal integration are

- Put logic as constraints on a DNN: a deep net is extended with an extra regularization term derived from some logical property or from extra heavy annotations. But it is not guaranteed that such a DNN can make consistent symbolic prediction, as suggested by Xu et al. [2017] that deep nets trained with additional logical regularizations cannot consistently make predicions that is from the logic they were trained on.
- Enable a DNN to execute logical programs Cohen et al. [2017]. We can make a logic program differentiable by using differentiable functions and let a neural network to execute the logical program, such as querying a database. These approaches predicts a transparent struture as output, but its prediction process is not transparent or following logical property at all.

Vertical integration, the category to which our Neural DNF belong, assembles a deep learning model and a symbolic model in a sequential manner. It intends to use deep learning for pattern recognition (perception), and the symbolic model for high-level reasoning. However vertical integration but does not recieve much attention even it has a strong neuroscience inspiration.

- SATNet by Wang et al. [2019a] opens a framework to design flexible combinatorial function in a smoothed differentiable way and the combinatorial relationship is learned instead of hard-coded grammar. But still, we do not have a control or a sense of understanding to interpret the prediction.
- Manhaeve et al. [2018] use DNN for perception and symbolic reasoning given a defined grammar . They use a DNN to handle perception and outputs a classification as a neural predicate, and then use this predicates to do reasoning. It learn a Probabilistic ProgLog program by gradient descent enabled by compiling a Sentential Decision Diagram. But the relationship of these predicates are hard-coded by the used grammar, not learned; and the symbols (that is, the extracted feature $\phi(x)$) is not learned from scratch.

Learning neural predicates that represent the abstract features/symbols ('cat's mouth', etc) and learning the relationship among these variables are a novel thing! And this is also very important if we wish to do symbolic prediction on raw unstrutured data. It is genrally true that most of the methods will need pre-training or heavy annotations of the symbols extracted by a DNN.

But note that for our Neural DNF both the features $\phi(x)$ and the rules g are learned from scratch. This is in fact quite non-trivial if we consider Neural DNF in the context of neuro-symbolic integration.

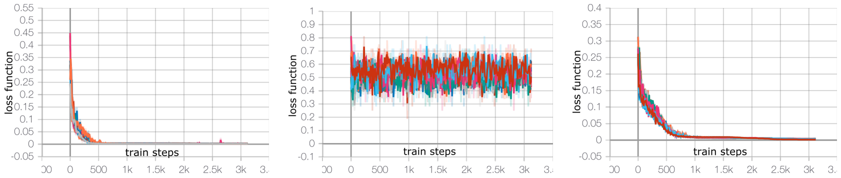
E Experiment

E.1 Evaluation of the BOAT Optimizer

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 [2019] 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. 1a we observe that multiple runs of BOAT give very similar and stable convergence, while in fig. 1b, 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. 1b).

There are some other issues with a large noise rate: (1) The slower learning of a larger noise rate can also be explained from the multiplicity nature of the DNF function we use (eq 2 and 3 in the main paper). This becomes more severe in high dimensional inputs. And finding a small noise rate initialization value will help. (2) when the noise rate intialized too high (and the hyperparameter of adaptivity rate not set right), it is also possible that the binary parameters cannot get huge gradient enough to flip, while it is only the continuous parameters including the noise rate are getting updated. This is an undesired result. In this case, find a smaller initilization rate.



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

Figure 1: Loss curve with 10 differently-generated synthetic dataset

Next, we apply BOAT and compared the convergence speed with the following baselines: (1) DNF-Real [Payani and Fekri, 2019, Wang et al., 2019b] and (2) DNF-STE which are discussed in **??**. (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. 2, 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.

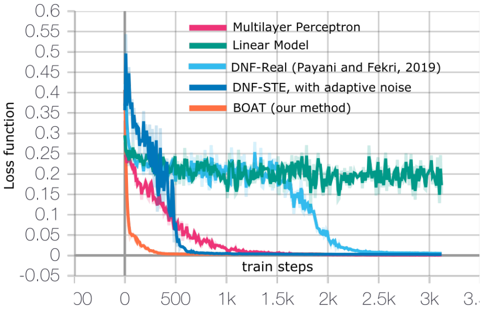


Figure 2: Loss Curve on the synthetic dataset

Here we evaluate only the second stage of our model, i.e., we use some datasets that the input features are already Boolean attributes, so we can use an identity function for the first stage neural module ϕ.

By removing ϕ, we focus on the evaluation of the learning algorithm, i.e., the proposed BOAT . We decide to use the synthetic Boolean bit-string dataset introduced by [Payani and Fekri, 2019] where this dataset is used for comparing the convergence of learning of the introduced DNF layer.

We randomly draw 5000 bit-strings where each bit is 0 or 1 (uniformly drawed). We then also randomly generate a ground-truth DNF and use this ground-truth DNF to assign labels to the bit-strings. Hopefully this ground-truth DNF should be recovered and indeed it is recovered. The ground-truth DNF consists of 5 clauses (rules) and each clause (rule) has 3 conditions. The conditions include negations. We random choose 4000 strings as the training set.

The generation of this synthetic Boolean bit-string dataset can be found at `dataloaders.py`. Note that in [Payani and Fekri, 2019], the bit is drawn with prob 0.75 to be zero and 0.25 to be one, and in our case we just use prob 0.5 to be zero and 0.5 to be one. This is because we also use negation in the generation of ground-truth DNF.

The DNF structure is 2K→N→1. We set $N = 64$ for the DNF as the default value for our method as well as DNF-Real and DNF-STE. For MLP and Linear model, we concatenate the input with its negation, and use a three-layer architecture (2K→N→1) for MLP, and the linear model is in essential a two-layer perceptron (2K→1). As the DNF layer uses negations, we will also do the same for the linear model and multi-layer perceptron(MLP). We simple concanate the input feature with its negations so that the linear model is of structure (2K→1) and the MLP (2K→N→1) For our method and all baselines we compare, we use Adam with initial learning rate 0.001. We use $\lambda_g = 0.0001$ as the default value.

Note that for compared baselines, if we use a larger learning rate such as 0.01 and since the synthetic Boolean bit-string dataset is not very difficult, the learning of DNF will be quicker (thus the covergence figure will looks different). However for our method, if we also set the initial learning rate for the noise from 0.001 to 0.01, the learning of BOAT will also be quicker (and a similar convergence figure should be able to be reproduced).

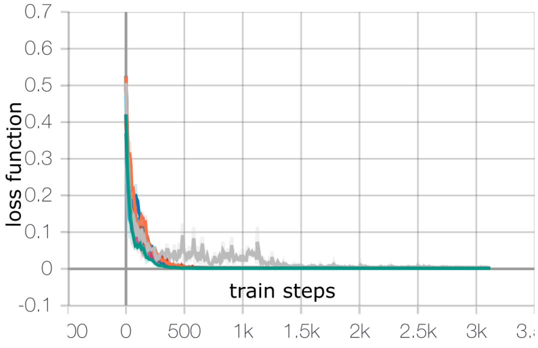


Figure 3: *Loss curve with 10 differently-generated synthetic dataset using zero-initialization*

random initialization and zero initialization. As we've mentioned in the main paper, one might suspect that the ground-truth DNF is not learned by our algorithm BOAT , but is only happened to be discovered by the random initialization. So here we also run our method with 10 different random seeds (differently-generated dataset) but with all-zero-initialization for $\boldsymbol{W}$ and $\boldsymbol{S}$. We can observe that the learning is still successful.

On real-world tabular datasets. In order to further investigate the performance, we also applied our method (learning the DNF g only) on several real-world datasets (table 1). We discretize the input features as preprocessing. Since these datasets are more complicated than the above synthetic dataset, we set N to 128; we set λ_g to 1e-4 as the default value. We find that our method perform competitively well and does not have performance decrease when the noise is removed. DNF-Real's performance slightly decreases after parameters are thresholded for Banknote dataset. One point worth noting is that learned DNF can achieve 100% F1-score on the tic-tac-toe dataset while weighted-sum-style models (linear Model/MLP/SENN[1]) can only approach to 100%. We think it is because the tic-tac-toe data is gathered from the status of a combinatorial game and thus may be more easily learned by rule-based models.

	DNF-BOAT With / Without noise	DNF-REAL Before / After thresholding	DNF-STE With / Without noise	Linear Model	MLP	SENN
Banknote	92.11%/92.11%	92.56%/91.05%	91.49%/91.49%	92.56%	92.11%	92.11%
tic-tac-toe	100%/100%	100%/100%	93.59%/93.59%	98.79%	99.59%	99.59%
Blogger	81.81%/81.81%	81.81%/81.81%	78.78%/78.78%	71.42%	86.95%	86.95%

Table 1: Test F1 score of learned DNF

Note that learning rule-based models using neural networks is already a challenging task, which we will not further investigate here.

E.2 The 2d-XOR

Four gaussians are drawn using scikit's 'make_blob' function with cluster mean (0,5), (5,0), (10,5) and (5,10).

E.3 MNIST

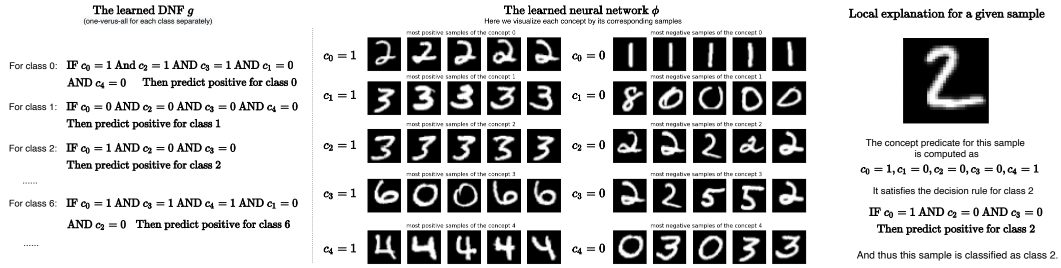
We use the standard train-test split for MNIST (in total 10000 test samples.)

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 $\boldsymbol{W}, \boldsymbol{S}$ for each class. After

---

[1] Note that SENN here is the second stage module of [Melis and Jaakkola, 2018] which is a MLP that takes input and generate the coefficients of a linear model.

training Neural DNF finds one or two rules for each class and can achieve over 99% test accuracy. 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 [Melis and Jaakkola, 2018]).


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 4: Explanations provided by Neural DNF on MNIST

Here we provide the explanations that Neural DNF derives: (1) In fig. 4a, 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. 4b, 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. 4b 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.

The detailed decision rule for each class is:

Decision rule for class 0 is IF c-0,c-2,c-3 = 1 AND c-1, c-4 = 0 THEN predict class 0

Decision rule for class 1 is IF = 1 AND c-0, c-2, c-3, c-4 = 0 THEN predict class 1

Decision rule for class 2 is IF c-0 = 1 AND c-2, c-3 = 0 THEN predict class 2

Decision rule for class 3 is IF c-0,c-1,c-2 = 1 AND c-3, c-4 = 0 THEN predict class 3

Decision rule for class 4 is IF c-3,c-4 = 1 AND c-0, c-2 = 0 THEN predict class 4

Decision rule for class 5 is IF c-1,c-2 = 1 AND c-0, c-3 = 0 OR c-0,c-1,c-2 = 1 AND c-0, c-1, c-2, c-4 = 0 THEN predict class 5

Decision rule for class 6 is IF c-0,c-3,c-4 = 1 AND c-1, c-2 = 0 THEN predict class 6

Decision rule for class 7 is IF c-0,c-1,c-3 = 1 AND = 0 THEN predict class 7

Decision rule for class 8 is IF c-0,c-2,c-4 = 1 AND c-3 = 0 OR c-2,c-4 = 1 AND c-1, c-3 = 0 THEN predict class 8

Decision rule for class 9 is IF c-1,c-2,c-3 = 1 AND c-0 = 0 THEN predict class 9

E.4 Other datasets in Section 4.2

We evaluate our method on datasets as follows: MNIST, KMNIST, SVHN, CIFAR10. These are not new but very standard image datasets and we just use standard train-test split comes with the pytorch backend. Note that the four datasets are image datasets so use an convolutional neural network.

We use a fixed adam learning rate and fixed γ for the modified *Bop* optimizer. So there is no learning rate decay or other scheduling. For MNIST and KMNIST we use 5 concept predicate and run for 100 epoch. For SVHN and CIFAR10 we use 32 concept predicate and run for 200 epoch as these two datasets are more challenging.

Note that we present results of Neural DNF using 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.). We can also use a random tie-breaking: when multiple classes are predicted as positive, we randomly pick one. We run 10 times on test set and report mean and standard deviation

Table 2: Test Accuracy of Neural DNF on some image datasets with two different tie-breaking

	MNIST	KMNIST	SVHN	CIFAR10
lazy tie-breaking	99.08%	95.43%	90.13%	67.91%
random tie-breaking	99.08% ± 0.02%	95.43% ± 0.03%	90.29% ± 0.04%	68.26% ± 0.12%

References

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.

David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.

Kay R Amel. From shallow to deep interactions between knowledge representation, reasoning and machine learning. In *Proceedings 13th International Conference Scala Uncertainity Mgmt (SUM 2019), Compiègne, LNCS*, pages 16–18, 2019.

P Arabshahi, TP Caudell, JJ Choi, and BJ Song. Steepest descent adaptation of min-max fuzzy if-then rules.

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.

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

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.

William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint arXiv:1707.05390*, 2017.

Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In *Advances in neural information processing systems*, pages 592–603, 2018.

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.

Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pages 524–532, 1990.

Fenglei Fan, Jinjun Xiong, and Ge Wang. On interpretability of artificial neural networks. *arXiv preprint arXiv:2001.02522*, 2020.

Hui Fen, Kuangyan Song, Madeilene Udell, Yiming Sun, Yujia Zhang, et al. Why should you trust my interpretation? understanding uncertainty in lime predictions. *arXiv preprint arXiv:1904.12991*, 2019.

Raquel Florez-Lopez and Juan Manuel Ramon-Jeronimo. Enhancing accuracy and interpretability of ensemble strategies in credit risk assessment. a correlated-adjusted decision forest proposal. *Expert Systems with Applications*, 42(13):5737–5753, 2015.

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

Johannes Fürnkranz, Tomáš Kliegr, and Heiko Paulheim. On cognitive preferences and the plausibility of rule-based models. *Machine Learning*, pages 1–46.

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.

Alexander L. Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *CoRR*, abs/1608.04428, 2016. URL `http://arxiv.org/abs/1608.04428`.

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`.

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.

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.

Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems*, pages 3781–3789, 2016.

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

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. In *International Conference on Machine Learning*, pages 2390–2399, 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.

Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. *arXiv preprint arXiv:1806.04854*, 2018.

Amir E Khandani, Adlar J Kim, and Andrew W Lo. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11):2767–2787, 2010.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

Himabindu Lakkaraju and Osbert Bastani. " how do i fool you?": Manipulating user trust via misleading black box explanations. *arXiv preprint arXiv:1911.06473*, 2019.

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.

Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detyniecki. The dangers of post-hoc interpretability: Unjustified counterfactual explanations. *arXiv preprint arXiv:1907.09294*, 2019.

Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

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`.

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.

Xiangming Meng, Roman Bachmann, and Mohammad Emtiyaz Khan. Training binary neural networks using the bayesian learning rule. *arXiv preprint arXiv:2002.10778*, 2020.

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

Hiroyoshi Nomura, Isao Hayashi, and Noboru Wakami. A learning method of fuzzy inference rules by descent method. In *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, pages 203–210. IEEE, 1992.

Petra Kralj Novak, Nada Lavrač, and Geoffrey I Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10(Feb):377–403, 2009.

Lennart Obermann and Stephan Waack. Demonstrating non-inferiority of easy interpretable methods for insolvency prediction. *Expert Systems with Applications*, 42(23):9117–9128, 2015.

Lennart Obermann and Stephan Waack. Interpretable multiclass models for corporate credit rating capable of expressing doubt. *Frontiers in Applied Mathematics and Statistics*, 2:16, 2016.

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

Andres Potapczynski, Gabriel Loaiza-Ganem, and John P Cunningham. Invertible gaussian reparameterization: Revisiting the gumbel-softmax. *arXiv preprint arXiv:1912.09588*, 2019.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

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.

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

Guolong Su, Dennis Wei, Kush R Varshney, and Dmitry M Malioutov. Interpretable two-level boolean rule learning for classification. *arXiv preprint arXiv:1511.07361*, 2015.

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.

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.

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. Or's of and's for interpretable classification, with application to context-aware recommender systems. *CoRR*, abs/1504.07614, 2015. URL `http://arxiv.org/abs/1504.07614`.

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.

Ning Xie, Gabrielle Ras, Marcel van Gerven, and Derek Doran. Explainable deep learning: A field guide for the uninitiated. *arXiv preprint arXiv:2004.14545*, 2020.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *arXiv preprint arXiv:1711.11157*, 2017.

Xin Zhang, Armando Solar-Lezama, and Rishabh Singh. Interpreting neural network judgments via minimal, stable, and symbolic corrections. In *Advances in Neural Information Processing Systems*, pages 4874–4885, 2018.