Neural Disjunctive Normal Form: Vertically Integrating Logic With Deep Learning For Classification

by

Jialin Lu

B.Eng., Zhejiang University, 2018

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Science

Approval

Name:	**Jialin Lu**
Degree:	**Master of Science (Computing Science)**
Title:	**Neural Disjunctive Normal Form: Vertically Integrating Logic With Deep Learning For Classification**
Examining Committee:	**Chair:** Valentine Kabanets Professor
	Martin Ester Senior Supervisor Professor
	Maxwell W. Libbrecht Supervisor Assistant Professor
	David Mitchell External Examiner Associate Professor
Date Defended:	**December 14, 2020**

Abstract

Inspired by the limitations of pure deep learning and symbolic logic-based models, in this thesis we consider a specific type of neuro-symbolic integration called *vertical integration* to bridge logic reasoning and deep learning and address their limitations. The motivation of vertical integration is to combine perception and reasoning as two separate stages of computation, while still being able to utilize simple and efficient end-to-end learning. It uses a perceptive deep neural network (DNN) to learn abstract concepts from raw sensory data and uses a symbolic model that operates on these abstract concepts to make interpretable predictions.

As a preliminary step towards this direction, we tackle the task of binary classification and propose the Neural Disjunctive Normal Form (Neural DNF). Specifically, we utilize a perceptive DNN module to extract features from data, then after binarization (0 or 1), feed them into a Disjunctive Normal Form (DNF) module to perform logical rule-based classification. We introduce the *BOAT* algorithm to optimize these two normally-incompatible modules in an end-to-end manner.

Compared to standard DNF, Neural DNF can handle prediction tasks from raw sensory data (such as images) thanks to the neurally-extracted concepts. Compared to standard DNN, Neural DNF offers improved interpretability via an explicit symbolic representation while being able to achieve comparable accuracy despite the reduction of model flexibility, and is particularly suited for certain classification tasks that require some logical composition. Our experiments show that BOAT can optimize Neural DNF in an end-to-end manner, i.e. jointly learn the logical rules and concepts from scratch, and that in certain cases the rules and the meanings of concepts are aligned with human understanding.

We view Neural DNF as an important first step towards more sophisticated vertical integration models, which use symbolic models of more powerful rule languages for advanced prediction and algorithmic tasks, beyond using DNF (propositional logic) for classification tasks. The *BOAT* algorithm introduced in this thesis can potentially be applied to such advanced hybrid models.

Keywords: neuro-symbolic integration; hybrid model; interpretability

Dedication

To *Liu Chang*.

Acknowledgements

I would like to thank my advisor, Martin Ester, for his enthusiasm and guidance throughout my master's study. It is not easy to figure out a difficult topic myself and explore it, and Martin is always supporting me.

I am a lucky man, and I am grateful for everything.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The tension between deep learning (connectionism) model and symbolic model is perhaps the most fundamental issue in the area of artificial intelligence (AI). During several decades of development and debate, it is generally agreed that there are pros and cons for both approaches.

In recent years, deep neural networks (DNN) has gained superior popularity and success in numerous domains. DNNs are highly flexible predictors. A DNN can be end-to-end optimized to capture complex mapping from raw sensory data to prediction, without manual feature engineering. The successes of DNN are mainly due to (1) efficient all-purpose end-to-end learning by backpropagation and (2) DNN's complex and overparameterized architecture that brings the flexibility to capture very complex patterns accurately. The power of end-to-end learning supports an efficient, elegant albeit somewhat brute-force way for the learning of DNN and has been demonstrated as essential in yielding state-of-the-art results [Krizhevsky et al., 2017]. Gradually pushing the limit of performance, DNN grows more and more complex.

Eventually, we result at *blackboxes* that are too complex for humans to understand: it is hard to interpret the meanings of hidden representations or how such representations are used for prediction. Furthermore, this lack of interpretability leads to difficulties for human to understand, interact with, or manipulate the model, when trying to debug undesired behaviors or to improve the model by incorporating human knowledge. While in several scenarios including vision [Bau et al., 2020] and natural language processing [Radford et al., 2017], it is shown that some of the hidden units correspond to human-interpretable concepts without being taught to do so, i.e., intermediate units in the hidden layers emerge to represent some high-level concepts such as sentiment without direct supervision, it still remains difficult to analyze whether and how such concepts are used for prediction by the following layers.

Symbolic models of AI, on the other hand, provide a more transparent and interpretable computation process, but are often too specialized and rigid, as well as lacking an efficient all-purpose learning algorithm [Hinton, 1990, Minsky, 1991]. Beyond the success on many well-defined problems, it is not flexible enough to solve many real-world ones. One key drawback of symbolic models, in particular, is that it requires input symbols *a priori* that can represent the attributes of data which are not available for many tasks. This means for raw sensory data like image or audio, symbolic models are simply not applicable unless some preprocessing is provided.

A possible solution to address the limitations of deep learning and symbolic models, as discussed and promoted by many researchers recently [Marcus, 2020, Bengio, 2019, Yi et al., 2019, Mao et al., 2019, Hudson and Manning, 2019, Penkov, 2019, Watson AI Lab, 2020], is to develop hybrid neuro-symbolic models [Besold et al., 2017, Garcez et al., 2019]. Out of many different approaches to neuro-symbolic integration, in this thesis we investigate one of them called **Vertical Integration**.

Our motivation for vertical integration is (1) the two-stage neuro-then-symbolic design of vertical integration to provide more interpretability, and (2) still being able to utilize the efficient end-to-end learning to handle sensory data like images.

As categorized in a recent neuro-symbolic survey [Garcez et al., 2019], the term **vertical integration** refers to the assembly of a deep learning model and a symbolic model in a sequential manner: basically, the deep learning model handles *perception*, and the symbolic model handles *reasoning* over the perceived information. It utilizes deep learning models to extract high-level concepts from raw sensory data (which deep learning are good at), and utilizes symbolic models to reason about the high-level concepts (which symbolic models are good at) to make interpretable predictions. 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 $\hat{y} = f(x) = g(\phi(x))$. The first-stage ϕ is a neural network feature extractor and the second-stage g is the symbolic model processing the extracted concepts into final prediction. Vertical integration certainly 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 it is more important that from a practical perspective, vertical integration provides a solution to demystify the blackbox as two stages of computation, handled by different modules. Compared to pure deep learning models that purposely behave a monolithic blackbox, vertical integration sacrifices flexibility but gains in interpretability at modular level: we can inspect the meaning of neurally-extracted concepts which ideally should be aligned with human understanding and potentially reusable, and unlike pure DNN, this time we can understand exactly how these concepts are processed by the symbolic model easily. In summary, it enables easier understanding of the model, as well as enables easier human inspection, interaction and manipulation.

Despite that we like to have a two-stage modular pipeline of vertical integration, we also want the power of an all-purpose end-to-end learning algorithm. *The bitter lesson* [Sutton, 2019] implicates that 'general methods that leverage computation are ultimately the most effective'. Following this implication as well as the power of end-to-end gradient-based learning, we consider a simple gradient-based learning algorithm that can jointly optimize the DNN and the symbolic model end-to-end. This means that (1) we can learn the abstract concepts and the symbolic model from scratch and (2) we do not have to resort to a juxtaposed combination of learning algorithms like gradient-based optimization and combinatorial search, which are of dramatically different nature. Another advantage of such a simple and general algorithm is that, ideally this algorithm is not specific to a particular symbolic model, while combinatorial search algorithms are usually specific to the constraints of a particular symbolic model. A simple and general algorithm can potentially be widely applicable to many different integrated symbolic models.

Regarding applications, 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 thesis, we start 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.

As mentioned above, we seek to optimize the deep neural network 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. Figure 1.1 gives the overall workflow of the model architecture of Neural DNF and the learning algorithm BOAT. 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.

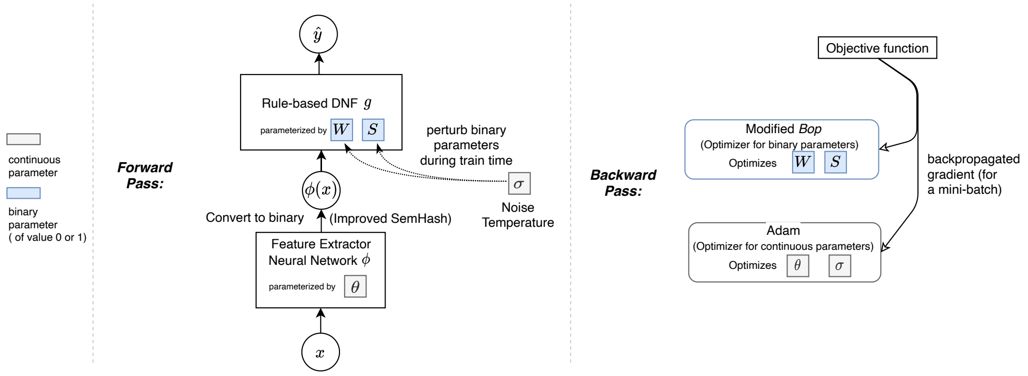

continuous parameter
binary parameter (of value 0 or 1)
Forward Pass:
$\hat{y}$
Rule-based DNF g
parameterized by W S
perturb binary parameters during train time
σ
Noise Temperature
$\phi(x)$
Convert to binary
(Improved SemHash)
Feature Extractor Neural Network ϕ
parameterized by θ
x
Backward Pass:
Objective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

Figure 1.1: *Overview of Neural DNF and the proposed BOAT Algorithm*

Compared to standard DNF, Neural DNF can handle prediction tasks from raw sensory data (such as images) thanks to the neurally-extracted concepts; compared to standard DNN, Neural DNF offers improved interpretability via an explicit symbolic representation while being able to achieve comparable accuracy despite the reduction of model flexibility. Our experiments show that it is possible for the end-to-end learning of Neural DNF to automatically obtain the correct logical rules and the abstract concepts whose meanings are aligned with human understanding.

The rule-based Neural DNF is particularly suited for classification tasks that (1) require the logical compositions of certain concepts and (2) the high-level concepts need to be learned from sensory data. We first 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 continue by applying Neural DNF on a newly-created dataset called MNIST-Sums-to-Odd. Each instance in this dataset consists two randomly-drawn digit images from MNIST and the label is defined as 1 if these two digits sum up to an odd number. Neural DNF can learn both concepts of abstract meaning and the logical rules at the same time. Specifically, the neurally-extracted concepts correspond to the abstract meaning of telling a digit image is an 'odd number' or not, and the learned logical rules correspond to a transparent computation procedure that is aligned with human understanding: *two digits sum up to an odd number only if one is an odd number and the other is an even number.* We consider MNIST-Sums-to-Odd as another good example of Neural DNF for the joint learning of concepts and logical rules from scratch. More than that, we believe learning the abstract concept of being 'odd' or 'even' as a by-product is non-trivial. 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 scenario 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 to that of blackbox deep learning models while offering an interpretable symbolic DNF representation, that makes human easier to interact with or manipulate the model.

In summary, we proposed Neural DNF and view it as an important first step towards more sophisticated vertical integration models in the future, which use symbolic models of more powerful rule languages for advanced prediction and algorithmic tasks, beyond using DNF (propositional logic) for classification tasks. The *BOAT* algorithm introduced in this thesis can potentially be applied to such advanced hybrid models.

1.1 Overview

This thesis investigates neuro-symbolic integration, specifically vertical integration, and presents an approach to vertical integration for the simplest prediction task: classification. The thesis touches upon many areas of machine learning that are not usually linked together: deep learning, symbolic models, hybrid models, interpretable machine learning, differentiable program induction, etc.

In Chapter 2, we introduce the background and related works that are essential for the understanding of the thesis. The first step in understanding the motivation of vertical integration of deep learning and symbolic logic is to understand (1) the limitations of pure deep learning and symbolic logic, discussed in Section 2.1, and (2) why it makes sense to combine these two (neuro-symbolic integration) and why specifically we combine them in a vertical way (vertical neuro-symbolic integration), discussed in Section 2.2.

In Chapter 3, we formulate the problem and our Neural DNF model.

In Chapter 4 we introduce our learning algorithm BOAT.

Chapter 5 contains the experiments for evaluating Neural DNF.

Chapter 6 concludes the thesis.

Chapter 2

Background

In this chapter, we provide the background for neuro-symbolic integration. Section 2.1 motivates the need for neuro-symbolic integration by discussing the complementary strengths and weaknesses of both learning approaches. Section 2.2 surveys methods in two categories of integration: horizontal integration and vertical integration.

2.1 Deep v.s. Symbolic Learning

The tension between deep learning (connectionism) model and symbolic model has been at the center of debate as one of the fundamental issues in the area of artificial intelligence (AI). For now, it is hard to draw a conclusion given the current state of development. But it is generally agreed that both the neural network, nowadays frequently coined as deep learning, and the more old-fashioned symbolic AI approaches have advantages and disadvantages in one's own.

In recent years, deep neural networks (DNN) have gained superior popularity and success in numerous domains. As there could be many reasons behind this success, we summarize two major ones as follow:

- Deep learning is able to utilize very complex and typically overparameterized architecture, which enables deep learning to capture very complex patterns accurately.
- Deep learning is able to utilize an efficient end-to-end learning algorithm by back-propagation, which is able to leverage the hardware such as GPU to perform quite scalable optimization.

Both the complex architecture and end-to-end learning have been demonstrated as essential in yielding state-of-the-art results [Krizhevsky et al., 2017]. Take a broader perspective, readers should be noted that both of these two reasons are inseparable from the other. Without the flexibility of complex architecture, deep learning will not have the capacity to model many difficult tasks; without an efficient learning algorithm, optimization of such massive parameters could have been a nightmare. Researchers used to consider learning with gradient-descent by backpropagation as limited, causing many problems like local minima and slow convergence, but magically these problems do not seem to get very severe with the ever-growing complexity in architecture, demonstrated by numerous achievements [Graves et al., 2014, 2016, Tamar et al., 2016, Mirowski et al., 2016]. As we can observe in the recent trend of deep learning research [Devlin et al., 2018], in order to push the boundary of state-of-the-art performance, the architecture of deep neural networks keeps growing more and more complex.

However, concerns about the interpretability of deep learning have been raised [Chakraborty et al., 2017], as we find deep neural networks to be *blackboxes* that are too complex for humans to understand. It is true that while the ability of deep learning on feature extraction and pattern recognition is widely recognized, it is certainly not easy for human to interpret the underlying mechanism. The lack of interpretability leads to difficulties for human to understand, interact with or manipulate the model when trying to account for trustworthy and safe decision making, debug undesired behaviors or to improve model by incorporating human knowledge. What is going on during so many layers of processing is simply overwhelming to understand. While in several scenarios including vision [Bau et al., 2020] and natural language processing [Radford et al., 2017], it is shown that some of hidden units correspond to human-interpretable concepts without being taught to do so, i.e., intermediate units in the hidden layers emerge to represent some high-level concepts such as sentiment without direct supervision, it still remains difficult to analyze whether and how such concepts are used for prediction by the following layers.

On the other hand, symbolic model of AI provides a more transparent and interpretable computation process, but are often too specialized and rigid, as well as lacking an efficient all-purpose learning algorithm [Hinton, 1990, Minsky, 1991]. Symbolic models of AI have achieved many successes on many well-defined problems, but it is not flexible enough to solve many real-world ones. One key drawback of symbolic models, in particular, is that it requires input symbols *a priori* that can represent the attributes of data. This means for raw sensory data like image or audio, symbolic model is simply not applicable unless some preprocessing is provided.

This leads to the question we are studying in this thesis: can we leverage the advantages of symbolic models, such as symbolic representation and interpretability, while still being able to handle some real-world data such as images?

2.2 Neuro-symbolic Integration

To reconcile the advantages of efficient learning in deep neural networks and reasoning and interpretability of symbolic representation, neural-symbolic integration has been an active topic of research for many years. Generally, under the umbrella terminology of neuro-symbolic integration [Besold et al., 2017], there are many different methods for combining neural methods with symbolic-logic methods, learning and reasoning. The vast literature on this topic offers a multitude of approaches covering different settings, making it difficult to discuss related works completely. Here we follow a most recent survey [Garcez et al., 2019] to introduce neuro-symbolic integration, which is divided into two main categories: Horizontal integration and Vertical integration.

2.2.1 Horizontal Integration

Horizontal integration aims at integrating neural and symbolic techniques into one inseparable model and most of the research works under 'neuro-symbolic integration' belong to this category. In an oversimplied manner, horizontal integration can either be accomplished by making neural model to behave more like a symbolic model, or making symbolic method more neural.

Option 1: Making neural model behaves more like a symbolic model. Also known as deep learning with logical constraints, it uses logical knowledge to improve neural network learning. To give an example, a simple way to do that is to extend a deep neural network with an extra regularization term derived from some logical property or from extra heavy annotations [Hu et al., 2016]. However, one should be noted that there is usually no guarantee for such a DNN to make consistent symbolic predictions. As suggested by Xu et al. [2017], DNNs trained with additional logical regularizations cannot consistently make predicions that are from the logic they were trained on.

Option 2: Making symbolic model more neural. this option converts the usual symbolic into something more like a neural network. A representative work can be TensorLog [Cohen et al., 2017], which can convert and compile logic programs into a differentiable computation graph. In this way, efficient inference can be computed in a more 'neural' fashion.

Our discussion for this category is oversimplified. Interested readers are encouraged to read [Besold et al., 2017].

2.2.2 Vertical Integration

The horizontal integration can be summarized by achieving some computation goals that can be done in one way in a different way, such as enabling a neural network to perform inference that used to be done by symbolic methods. Vertical integration, on the other hand, focuses on combining neural and symbolic models for something new that neither pure deep learning nor symbolic model is capable of. Vertical integration, to which the category to which our Neural DNF belong, assembles a deep learning model and a symbolic model in a sequential manner. Framed as a two-stage approach, it intends to use deep learning for pattern recognition (perception), and the symbolic model for high-level reasoning. This specific way of integration provides benefits compared with either pure neural or pure symbolic models. On the one hand, the symbolic model is extended so to be able to work on raw sensory data by the interface of neurally-extracted features; On the other hand, the deep neural network is extended so to have a high-level symbolic module that is more transparent and interpretable.

Vertical integration is strongly inspired by biological observations: 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].

While the first-stage neural network can be simply viewed as a generic feature extractor, the choice of which symbolic model to integrate is the central problem in vertical integration. Unsurprisingly, the choice of symbolic model to integrate depends on the applied task. Here we introduce several representative works. 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* given pixel input; 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. It learns a Probabilistic ProgLog program by gradient descent enabled by compiling a Sentential Decision Diagram. Gaunt et al. [2016, 2017] provide a more general and accessible set of symbolic models as customizable differentiable procedural programs. A program template is hand-written with learnable components such as control loops and neural network modules. Then a gradient-based optimization algorithm learns both the program and the neural network in an end-to-end manner. In this thesis, we begin with the most basic task of binary classification, so we choose propositional logic in Disjunctive Normal Form (DNF), also known as 'decision rules' or 'rule set'. It is worth noting that although Gaunt et al. [2016, 2017], Manhaeve et al. [2018] provides a general toolkit, it does have some constraints on the learnable components within the program template, and learning a Disjunctive Normal Form like our work is not supported.

The key technical challenge of vertical integration, however, is how to learn such hybrid model. To a certain degree, the learning of vertical integration is like a chicken-egg problem. It is perhaps unsurprising that we are able to successfully learn the neural network given the ground-truth second-stage symbolic model, or learn the symbolic model given the ground-truth first-stage neural network very easily [Penkov, 2019]. But learning the neural and symbolic modules together from scratch is not trivial. Despite our proposal to optimize the neural and symbolic model from scratch together, there exists alternative approaches that combines existing combinatorial search algorithms for symbolic model with the gradient-descent optimization for the neural networks. A straightforward approach is treating the learning of a symbolic program as an iterative combinatorial search and during each search step, we fix the symbolic model and train the neural network. In each step, a possible configuration of the symbolic model is proposed and constructed, and then given this symbolic model fixed, the neural network is trained with gradient descent from scratch to convergence. The resulted model is evaluated and new configurations of the symbolic model can be proposed. This process is iterated until some termination criteria are met. As demonstrated by [Valkov et al., 2018], this approach is certainly generally-applicable because it can utilize the existing combinatorial search algorithms for the symbolic model, but comes with expensive computational cost, as each step requires a full training session of the neural network. To accelerate the learning, we suspect some alternating 'wake-sleep' learning or EM-like algorithm can be potentially applied. In our opinion, such a combination of algorithms seems too complex, and thus we start to consider a simple learning algorithm. In contrary to this juxtaposed combination of different algorithms, we in this thesis tackle the learning problem with a end-to-end gradient-based algorithm. We consider developing a gradient-based algorithm that can jointly learn the neural network and the symbolic model together from scratch.

Chapter 3

Neural Disjunctive Normal Form

In this chapter we introduce the proposed Neural Disjunctive Normal Form (Neural DNF). We start with the overall problem setting and definition in Section 3.1, then present the symbolic part (Section 3.2) and the neural part (Section 3.3) of Neural DNF respectively, and finally propose the objective function of Neural DNF in Section 3.4.

3.1 Problem Setting and definition

We consider a standard supervised learning setting of classification: given a dataset of $D = \{(x, y)\}$, 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 $\hat{y} = f(x) = g(\phi(x))$.

3.2 The symbolic DNF module

The second stage g is a logical rule-based symbolic model, formulated as a Disjunctive Normal Form (DNF). g takes a set of Boolean predicates as input feature and makes binary prediction.

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 clause (AND) of DNF can be viewed as a 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 an equivalent DNF and thus any rule-based binary classifier including decision set/list/tree can be expressed as a DNF.

A DNF 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: $r_1 \vee r_2 \vee \ldots \vee r_n$.

Learning of 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 K input concept predicates $\boldsymbol{c} = \{c_1, c_2, \ldots, c_K\}$, g computes the Boolean function as:

$$\hat{y} = g(\boldsymbol{c}) = \bigvee_{\boldsymbol{S}_j=1}^{N} \bigwedge_{\boldsymbol{W}_{i,j}=1} b_i \qquad \text{where } \{b_1, b_2, \ldots, b_{2K}\} = \{c_1, \neg c_1, c_2, \neg c_2, \ldots, c_k, \neg c_k\} \tag{3.1}$$

Note that eq. (3.1) is used during training; after training, $\boldsymbol{W}_{2K\times N}$ is discarded and only the selected rules are stored.

In some recent state-of-the-art algorithms [Lakkaraju et al., 2016, Wang et al., 2017], the learning algorithm of DNF consists of two phases: in the first phase, a fixed $\boldsymbol{W}$ is constructed by rule pre-mining; and then in the second phase, the actual '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. 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 S is not compatible with gradient-based optimization.

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 (initialized randomly); (II) we introduce a differentiable replacement of the logical operation of eq. (3.1) so that gradients can be properly backpropagated:

$$r_j = \bigwedge_{\boldsymbol{W}_{i,j}=1} b_i \quad \xrightarrow{\text{replaced by}} \quad r_j = \prod_{i}^{2k} \boldsymbol{F}_{\text{conj}}(b_i, \boldsymbol{W}_{i,j}), \text{ where } \boldsymbol{F}_{\text{conj}}(b, w) = 1 - w(1-b) \tag{3.2}$$

$$\hat{y} = \bigvee_{\boldsymbol{S}_j=1} r_j \quad \xrightarrow{\text{replaced by}} \quad \hat{y} = 1 - \prod_{j}^{N}(1 - \boldsymbol{F}_{\text{disj}}(r_j, \boldsymbol{S}_j)), \text{ where } \boldsymbol{F}_{\text{disj}}(r, s) = r \cdot s \tag{3.3}$$

eqs. (3.2) and (3.3) computes the DNF function exactly as eq. (3.1), 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 thesis, 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).

At last, it is worth mentioning that obtaining the differentiable replacement of eqs. (3.2) and (3.3) 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].

3.3 The neural network module

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. We use the term of concept *predicate* here to emphasize that this variable has a Boolean interpretation: it indicates the presence or the absence of a concept. As ϕ is parameterized by a neural network θ, the neural network ϕ's output $\tilde{c_i}$ is real-valued. Here we assume $\tilde{c_i}$ lies in the range of $[0, 1]$, which can be done simply by applying a sigmoid function. 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 $\tilde{c_i} > 0.5$ and $c_i = 0$ otherwise. However, since gradient through this step function is zero almost anywhere and thus prevents training, we utilize the *Improved SemHash* [Kaiser and Bengio, 2018] (which will be introduced later). 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'. For simplicity, without further notice we do not elaborate on ϕ and only assume ϕ to be some generic feedforward neural network.

3.3.1 Binarization into concepts: Improved SemHash

The feature extractor ϕ processes the raw input x into a set of intermediate representations $\{c_1, c_2, \dots, c_k\}$ called concept predicates. As ϕ is parameterized by a neural network θ, ϕ's output $\tilde{c_i}$ is real-valued. Assuming $\tilde{c_i}$ lies in the range of $[0, 1]$ (after applying sigmoid), we use a binary step function to discretize $\tilde{c_i}$ into Boolean predicates:

$$c_i = \mathbf{1}(\tilde{c_i}) = \begin{cases} 1, & \text{if } \tilde{c_i} > 0.5 \\ 0, & \text{otherwise.} \end{cases}$$

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 draws Gaussian noise ϵ with mean 0 and standard deviation 1. After the noise ϵ is added to $\tilde{c}$, two vectors c and c' are then computed. The first vector c is computed by applying the binary step function:

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

The second vector c' is computed by the following function called saturating sigmoid function [Kaiser and Sutskever, 2015, Kaiser and Bengio, 2016]:

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

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 $\tilde{c}$. Specifically, 'half-the-time' means 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. During testing, the noise is disabled and c is used as output.

The use of saturating sigmoid function on computing the second vector c', instead of just sigmoid function, is introduced first by a previous work Kaiser and Sutskever [2015] which claims that it has slight improvement. But we do not observe such difference in Neural DNF, so in our implementation we simply computed

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

Basically we use the simpler sigmoid function instead of the saturating sigmoid function.

We name two reasons for 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 several alternatives that need tuning of annealing, including the annealed sigmoid/tanh [Bulat et al., 2019] and gumbel-softmax trick [Maddison et al., 2016], but 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.

3.4 Objective function

The overall 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. In this thesis we use MSE for the classification loss by default.

For the regularization of g, we simply 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.

Chapter 4

The *BOAT* algorithm for learning Neural DNF

In this chapter, we introduce Bi-Optimizer learning with Adaptively-Temperatured noise (BOAT), the algorithm for learning Neural DNF. We note that the BOAT algorithm is a general algorithm, that is not specific to only Neural DNF, and can be potentially be applied to other models such as more sophisticated vertical integration models or models with mixed continuous-discrete parameters.

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 called *Bop* adopted from [Helwegen et al., 2019] to optimize the binary parameters $\{\boldsymbol{W}, \boldsymbol{S}\}$ of the DNF g. In Section 4.1, we introduce the motivation of BOAT. In Section 4.2, we present the modified Bop optimizer. Section 4.3 presents the concept of adaptively temperatured noise, a 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. (4.1)). We emphasize that (1) the introduced noise is necessary as otherwise learning of W,S constantly fails even in very simple cases (section 5.1) 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. Section 4.4 summarizes the overall algorithm and provides a pseudocode

4.1 Motivation

The motivation for developing our new algorithm BOAT is based on the observation of limitations of existing methods.

First, 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.

There are, however, two categories of existing methods that can be applied alternatively for this goal. 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.

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

4.2 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. In this section we begin with the introduction of *Bop* [Helwegen et al., 2019] and then introduce how it is minorly modified to suit our need.

Helwegen et al. [2019] propose the *Bop* optimizer in the context of binarized neural networks of value $\{-1, 1\}$. *Bop* [Helwegen et al., 2019] provides a new perspective on training binary neural networks without the need of the so-called latent weight like the aforementioned STE. Unlike updating latent weight by gradients and flip the parameter when the latent weight cross threshold value, it directly optimizes parameters of discrete value and flips the parameters only it receives a consistent gradient signal. 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

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

We make minor modifications to suit *Bop* (originally for $\{-1, 1\}$) into the case of $\{0, 1\}$. The Modified *Bop* optimizer introduced in this thesis 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.

4.3 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. (3.2) and (3.3) 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:

$$\tilde{w} = \begin{cases} 1 - \sigma_w \cdot \epsilon & \text{if } w = 1 \\ 0 + \sigma_w \cdot \epsilon & \text{if } w = 0 \end{cases}, \text{ where } \epsilon \sim \text{Uniform}(0, 1) \qquad (4.1)$$

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 every 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 (fig. 5.1c) , and σ cannot be too small: in the extreme case with zero noise the optimization of $\boldsymbol{W}, \boldsymbol{S}$ will constantly fail (fig. 5.1b) . 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 can also apply the same noise for DNF-STE, but it converges slower (fig. 5.2) . 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 the main advantage of *Bop*.

4.4 Overall Algorithm of BOAT

Here we provide pseudocode of the overall BOAT algorithm.

Using BOAT, the parameter $g_{\boldsymbol{W},\boldsymbol{S}}$ of the symbolic DNF module can be optimized jointly with the neural module ϕ_θ. What is important here is that BOAT is a simple and general learning algorithm for Neural DNF and potentially other hybrid models with mixed binary-continuous parameters. BOAT requires only minimal changes to the standard deep learning workflow. Ideally, this will improve the general applicability of the BOAT algorithm.

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 $\sigma_w$ (eq. (4.1)).
6:     Use $\theta$ and perturbed $\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}$ to compute the objective $\sum_{x_i, y_i} \boldsymbol{L}(g_{\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{S}}}(\phi_\theta(x_i)), y_i)$
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 $\sigma_w = min(0.5, max(\sigma_w, 0))$
```

Chapter 5

Experiments

In this chapter, we present the experimental evaluation of Neural DNF, which can be roughly divided into the following parts.

(1) In Section 5.1, we first evaluate the BOAT algorithm for learning the symbolic DNF alone. This serves as the basic testbed and sanity check for the BOAT algorithm. Specifically, we demonstrate the superior convergence of BOAT and the necessity of the adaptively-temperatured noise, the key ingredient of BOAT.

(2) In Section 5.2, we report results on a toy dataset called 2D-XOR, a minimal example to test Neural DNF and demonstrate that it can learn both learn the right high-level features (concepts) from the raw data and the right XOR function of these concepts to accurately predict the class label.

(3) In Section 5.3, we evaluate Neural DNF on a newly-created dataset called MNIST-Sums-to-Odd. It can be seen as a more advanced test case, with a learned concept that is of more abstract meaning than the concepts in 2D-XOR.

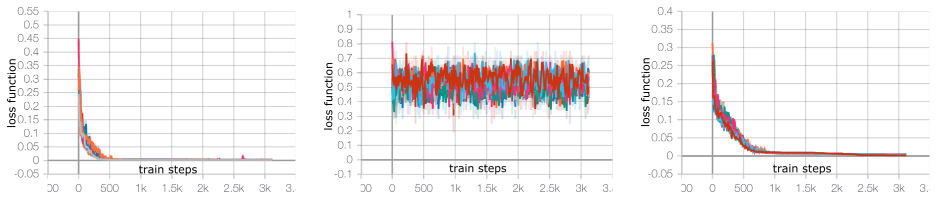
(4) We further apply Neural DNF to image datasets in two scenarios in Section 5.4 and Section 5.5: In the first scenario, we use a regular deep neural network as feature extractor with no further constraints. We show that BOAT can successfully learn both the feature extractor and the logical rules for classification, and achieve competitive accuracy. Note that in this case there is no guarantee that the extracted features are meaningful to a 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, and the interpretability enables a 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.

5.1 Evaluation of the BOAT Optimizer with the second stage alone

What comes first is to evaluate the proposed BOAT with its alternatives. 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.

We randomly draw 5000 bit-strings where each bit is 0 or 1 (uniformly drew). We then also randomly generate a ground-truth DNF and use this ground-truth DNF to assign labels to the bit-strings. The ground-truth DNF consists of 5 clauses (rules) and each clause (rule) has 3 conditions. The conditions include negations. We randomly choose 4000 strings as the training set and the rest 1000 strings as test set.

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. 5.1a we observe that multiple runs of BOAT give very similar and stable convergence, while in fig. 5.1b, 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. 5.1b).



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

Figure 5.1: 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 chapter 4. (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.

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 essence 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 concatenate 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 is the default value.

As shown in fig. 5.2, 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.

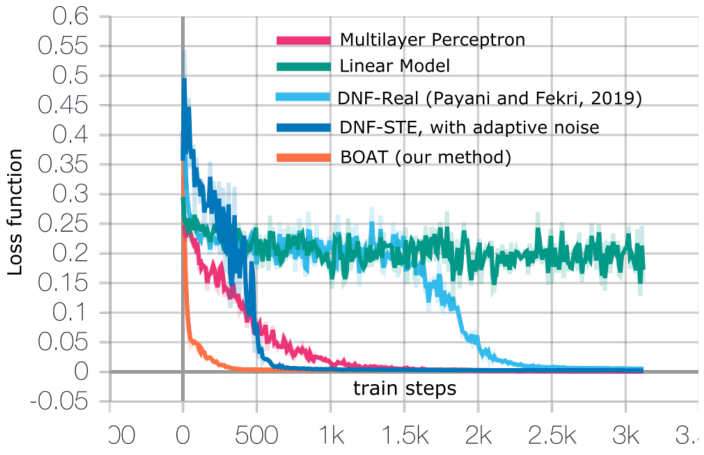


Figure 5.2: Loss Curve on the synthetic dataset

As for performance, all the above methods except the 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.

Note that since we use random initialization for $\boldsymbol{W}$ and $\boldsymbol{S}$, 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 run our method with 10 different random seeds (differently-generated

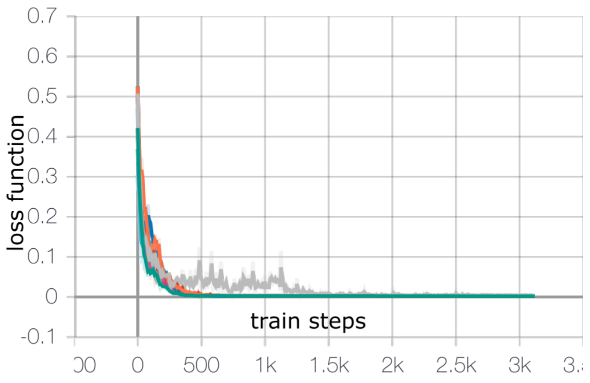


Figure 5.3: *Loss curve with 10 differently-generated synthetic datasets using zero-initialization*

dataset) but with all-zero-initialization for $\boldsymbol{W}$ and $\boldsymbol{S}$. In fig. 5.3, we can observe that the learning is still successful and very similar to the randomly-initialization one (fig. 5.1a).

5.2 Evaluation on 2D-XOR: a 2D toy dataset.

Here we apply Neural DNF on a 2D toy dataset 2D-XOR (fig. 5.4). 2D-XOR is generated by first drawing four isotropic 2D Gaussian clusters (drawn using scikit's 'make_blob' function with cluster mean (0,5), (5,0), (10,5) and (5,10)) and mark them as red square, blue square, blue circle and red circle (in clockwise order). We use an XOR-like label assignment, labeling red squares and blue circles as positive and the rest two clusters as negative. We consider

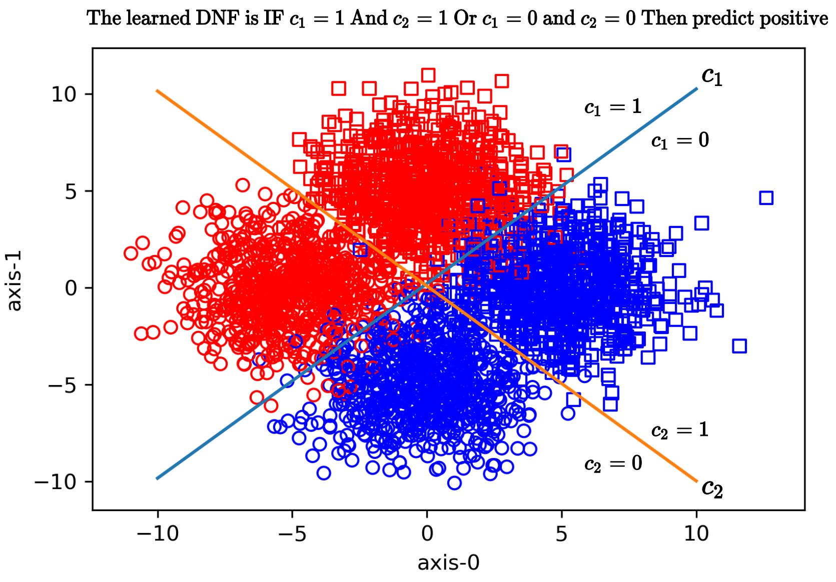


Figure 5.4: Neural DNF on 2D-XOR

2D-XOR as an extension of the historically important XOR problem [Minsky and Papert, 1969] that once causes an AI winter. 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 be visualized 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. 5.4, 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. 5.4 as a minimal working example of Neural DNF demonstrating that it is useful when (1) the features 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.

5.3 Evaluation on MNIST-Sums-to-Odd

While the above 2D-XOR dataset should serve as a good example of why jointly learning of the concept predicate and DNF rules is useful, it still seems to be trivial, as it assumes a simple XOR function to be learned and that the concepts are basically a linear separator in 2D space. Here we consider an advanced case where the concepts that have more abstract meaning.

We create the MNIST-Sums-to-Odd dataset. Each sample in MNIST-Sums-to-Odd consists of two MNIST digits, and the corresponding label is defined as 1 if these two digits sum to be an odd number, and 0 otherwise. Note that digit images of the MNIST-Sums-to-odd For the training set, we randomly draw 10000 pairs of digit images from the original MNIST training set, and for test set 10000 pairs of digit images from the original MNIST test set, ensuring none of the digit images in the test set have ever appeared during training.

For applying Neural DNF on this dataset, we choose the feature extractor to be a CNN that has only one output node (produce one feature predicate), and we apply the same CNN twice on each of the two digit images. Specifically, we adopt a LeNet-like architecture, but with the modfications as follows: (1) we modify the final layer so to have only one output node. (2) the output nodes are binarized (training with the Improved SemHash). Since we apply the CNN twice on two digit images, the two output nodes are fed into the DNF as concept predicates.

The learned Neural DNF model achieves a 98.03% test accuracy. Note that prediction accuracy is not the sole goal, we here look deeper into the inner structure of the Neural DNF. As the ground truth of MNIST-Sums-to-odd is constructed by checking whether the two digits sum up to an odd value, one can anticipate that a natural way that human can solve this problem in logic can be expressed as: *one of the two digits is an even number and the other is an odd number*. Following this expectation, we inspect the learned Neural DNF and found that first, the learned DNF rules match exactly the above logic, and more importantly, we find that the concept predicate corresponds exactly to the abstract meaning of being odd or even, by the accuracy of 99.06% (fig. 5.5). This means the feature extractor learns to correctly predicate the abstract meaning of being odd or even for the digit image. We emphasized that learning this abstract concept is not trivial, as (1) this abstract meaning of being odd or even is not given as supervision information, but rather an indirect by-product of joint learning, and (2) being odd or even is a very abstract meaning that is much harder than the 2D space linear separator in the previous 2D-XOR dataset.

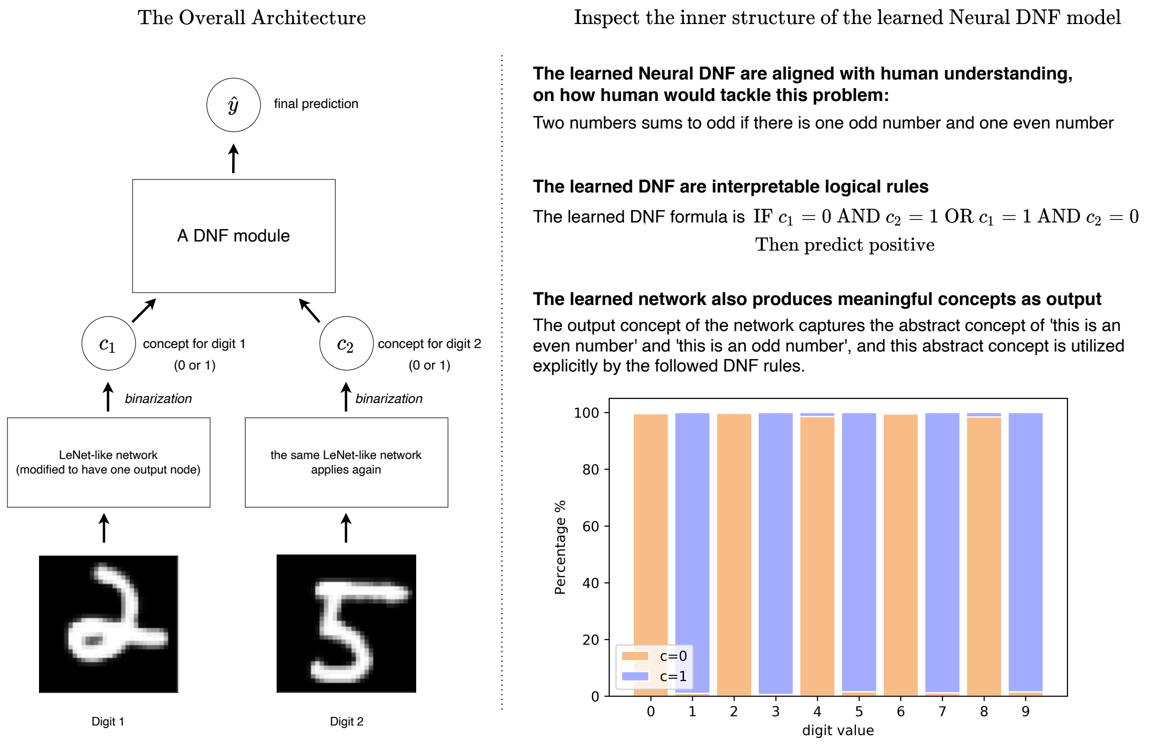

The Overall Architecture
Inspect the inner structure of the learned Neural DNF model
$\hat{y}$
final prediction
A DNF module
c_1
concept for digit 1 (0 or 1)
c_2
concept for digit 2 (0 or 1)
binarization
binarization
LeNet-like network (modified to have one output node)
the same LeNet-like network applies again
Digit 1
Digit 2
The learned Neural DNF are aligned with human understanding, on how human would tackle this problem:
Two numbers sums to odd if there is one odd number and one even number
The learned DNF are interpretable logical rules
The learned DNF formula is IF $c_1 = 0$ AND $c_2 = 1$ OR $c_1 = 1$ AND $c_2 = 0$ Then predict positive
The learned network also produces meaningful concepts as output
The output concept of the network captures the abstract concept of 'this is an even number' and 'this is an odd number', and this abstract concept is utilized explicitly by the followed DNF rules.
Percentage %
digit value
c=0
c=1

Figure 5.5: *Neural DNF on MNIST sums to odd.* the neural network part of the Neural DNF model can achieve F1 score with 99.07% and accuracy of 99.06% for predicting the abstract meaning of 'being odd or even' as a by-product.

We also evaluate and compare with some other alternative baselines for this dataset (table 5.1). We first adopting the same design of the feature extractor that applies on two images separately (but without the binarization) and replace the DNF with a linear model or a multi-layer perceptron (MLP). We also vary the number of output nodes of the feature extractor. We find that using linear model as the second stage classifier with either 1-dimension or 100-dimension embeddings does not result in a good generalization in terms of prediction accuracy, as it can achieve relatively high train-time accuracy but give almost random guess in test time. We argue that this may be because linear model cannot handle the non-linearity of the second stage function even if the first stage is a deep neural network, and that the DNN somehow learns to memorize all the training samples but learns no generalizable knowledge. This can be supported by our observation that the training of CNN (1-d) + linear model and CNN (100-d) + linear model only begin to deviate from random guess when we apply a very small learning rate and that it takes much more epochs to slowly achieve training accuracy above 90%.

When we replaced the linear model with a multi-layer perceptron, we can start to observe very decent test accuracy, that is comparable to our Neural DNF model. We believe this is because a multi-layer perceptron (MLP) is a universal approximator that can handle any non-linear functions. Beyond the prediction accuracy, we also inspect the embedding space of its feature extractor and find that the samples of even and odd digit images also seem to be separable in the embedding space (by applying some post-hoc dimension reduction and thresholding). In the 1-d embedding case, we can find an optimal threshold value that achieves 97.57% accuracy in predicting ‘being even or odd’, slightly lower than 99.06% of that from our Neural DNF model (which requires no post-processing for the concepts). But unlike Neural DNF, whether this information of separating ‘even and odd’ is clearly used by the MLP, and whether the MLP computes a function that is similar to the DNF rules remain in question.

Lastly, we consider a complete blackbox DNN that takes the concatenated two images together to predict the label. We find that it achieves the highest training accuracy but does not generalize well (95% test accuracy). In this case, however, the entire model is a non-interpretable blackbox.

Table 5.1: Accuracy on MNIST-Sums-to-Odd

	Train Acc	Test Acc
Neural DNF	99.33%	**98.03%**
CNN (1-d) + linear model	93.28%	50.12%
CNN (100-d) + linear model	94.56%	50.46%
CNN (1-d) + MLP	99.62%	97.72%
CNN (100-d) + MLP	99.84%	97.80%
Blackbox	**99.87%**	95.48%

To summarize, we view MNIST-Sums-to-Odd as an advanced example to demonstrate why the end-to-end optimization of perception feature extractor and logical rules together is useful, and that Neural DNF in this case can learn both the DNF logical rules and abstract concepts that are interpretable and make sense to human.

5.4 Evaluation on image datasets, scenario 1

In this section we apply Neural DNF with a CNN as feature extractor without any further constraints. and then demonstrate empirically that given the same first-stage feature extractor architecture, Neural DNF sacrifices slight loss of accuracy (table 5.2) while gaining more faithful interpretations (fig. 5.7).

In table 5.2, we first evaluate the accuracy of Neural DNF on several image datasets MNIST, KMNIST, SVHN, CIFAR10 using the standard train-test split. For MNIST and KMNIST we use 5 concept predicate and run for 100 epochs. For SVHN and CIFAR10 we use 32 concept predicate and run for 200 epochs as these two datasets are more challenging.

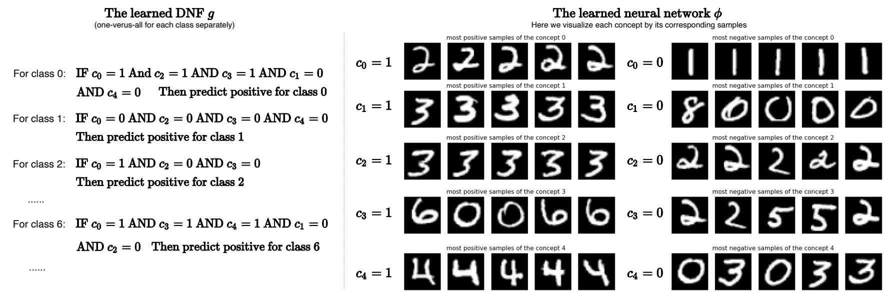
Using the same architecture for ϕ, the models we compare are (1) Neural DNF (2) a standard DNN (i.e. a linear model as g) and (3) self-explaining neural network (SENN) [Melis and Jaakkola, 2018], which uses a neural network to generate the coefficients of a linear model conditioned on the input and claims to have better interpretability. For MNIST and KMNIST we use 5 concept predicate and run for 200 epochs. For SVHN and CIFAR10 we use 32 concept predicate and run for 200 epocsh as these two datasets are more challenging. The same number of concept nodes are used for the baselines so to be a fair comparison. 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 setting 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 5.2: Test Accuracy on some image datasets

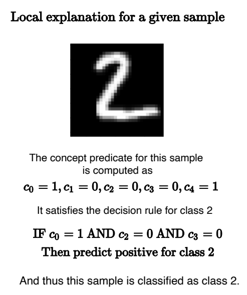
	MNIST	KMNIST	SVHN	CIFAR10
Neural DNF	99.08%	95.43%	90.13%	69.83%
standard DNN (with Binarization)	99.12%	95.86%	90.74%	70.43%
standard DNN (without Binarization)	99.11%	96.02%	91.45%	71.45%
SENN (with Binarization)	98.48%	92.64%	90.79%	71.09%
SENN (without Binarization)	98.50%	91.46%	92.15%	72.32%

We provide an anecdotal example on the explanations Neural DNF can derive in (fig. 5.6) as a direct comparison to the explanations provided by linear models as the second state (e.g., the MNIST example from the self-explaining network [Melis and Jaakkola, 2018]). We show the learned rules of DNF as well as maximum/minimum activated samples for each concept predicate. This can be viewed as explanations of what these concepts mean.

Now regarding the interpretability, the problem here is, we can in principle inspect 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 represen-


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
$c_4 = 1$
$c_4 = 0$

(a) Global explanation for the overall Neural DNF Model


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.

(b) Local explanation

Figure 5.6: Explanations provided by Neural DNF on MNIST

tations is still non-interpretable. We will revisit this interpretable representation problem in the next section.

Set aside the first-stage, here we can still compare the interpretability of the second-stage model. Despite many qualitative arguments for favoring the interpretability of the symbolic DNF [Rudin, 2019, Freitas, 2014, Huysmans et al., 2011, Wachter et al., 2018], we can further quantitively evaluate the faithfulness of models' interpretability, a critical criteria that measures how faithful the explanation for a particular prediction is 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.[1] We report the faithfulness metric on test set for DNN, SENN and Neural DNF in fig. 5.7. For DNN, we use the coefficients of the final linear layer as relevance scores. Alter-

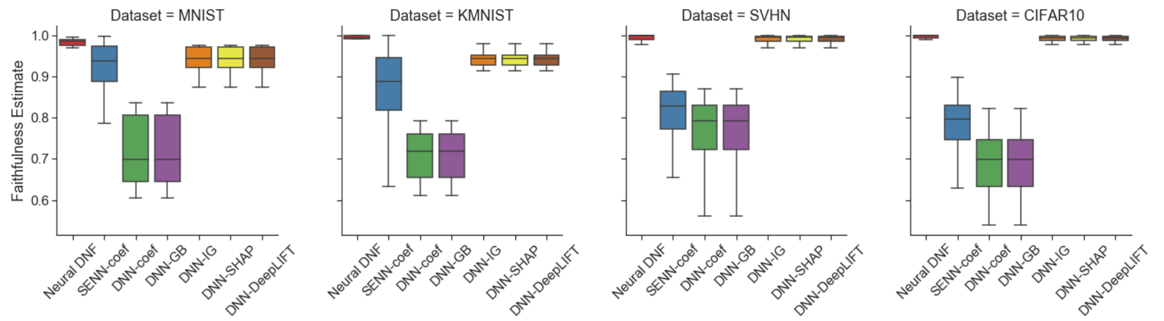


Figure 5.7: Evaluating the faithfulness of explanations on test set

natively, for DNN we can also utilize some representative post-hoc interpretation methods (also known as attribution methods): Guided Backprop (GB) [Springenberg et al., 2014], gradient SHAP [Lundberg and Lee, 2017], Integrated Gradient (IG) [Sundararajan et al.,

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

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. 5.7, 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.

5.5 Evaluation 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 classes and 112 binary annotated 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 an Inception-V3 based feature extractor ϕ and a linear layer g. Given concepts annotations, a concept prediction loss can be applied so that the extracted features 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 counterintuitive, 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 independently 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 extract interpretable features without the need of annotations can solve it, in that case joint training should give better results. In table 5.3 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 as other models do. But after intervention both Neural DNF and the concept bottleneck model achieve perfect accuracy. This indicates that (1) the accuracy by applying human intervention can be improved significantly which is the benefit of having a 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 5.3: 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

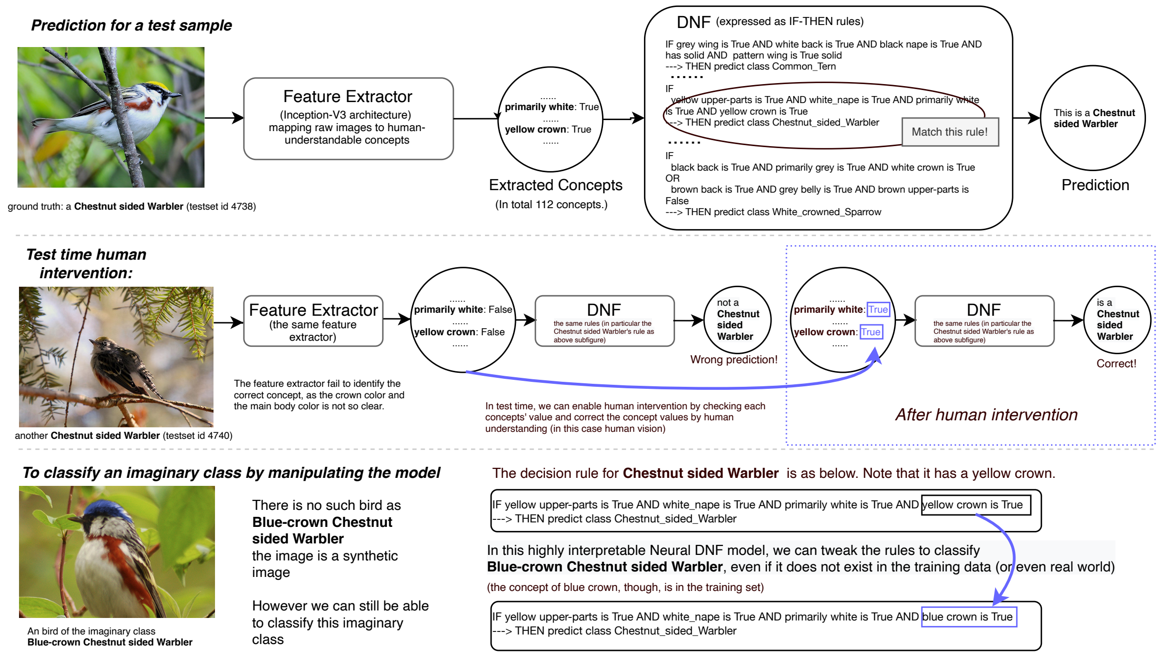

Prediction for a test sample
ground truth: a Chestnut sided Warbler (testset id 4738)
Feature Extractor
(Inception-V3 architecture)
mapping raw images to human-understandable concepts
......
primarily white: True
......
yellow crown: True
......
Extracted Concepts
(In total 112 concepts.)
DNF (expressed as IF-THEN rules)
IF grey wing is True AND white back is True AND black nape is True AND has solid AND pattern wing is True solid
---> THEN predict class Common_Tern
......
IF
yellow upper-parts is True AND white_nape is True AND primarily white is True AND yellow crown is True
---> THEN predict class Chestnut_sided_Warbler
Match this rule!
......
IF
black back is True AND primarily grey is True AND white crown is True
OR
brown back is True AND grey belly is True AND brown upper-parts is False
---> THEN predict class White_crowned_Sparrow
This is a Chestnut sided Warbler
Prediction
Test time human intervention:
another Chestnut sided Warbler (testset id 4740)
Feature Extractor
(the same feature extractor)
......
primarily white: False
......
yellow crown: False
......
DNF
the same rules (in particular the Chestnut sided Warbler's rule as above subfigure)
not a Chestnut sided Warbler
Wrong prediction!
The feature extractor fail to identify the correct concept, as the crown color and the main body color is not so clear.
In test time, we can enable human intervention by checking each concepts' value and correct the concept values by human understanding (in this case human vision)
......
primarily white: True
......
yellow crown: True
......
DNF
the same rules (in particular the Chestnut sided Warbler's rule as above subfigure)
is a Chestnut sided Warbler
Correct!
After human intervention
To classify an imaginary class by manipulating the model
An bird of the imaginary class
Blue-crown Chestnut sided Warbler
There is no such bird as Blue-crown Chestnut sided Warbler
the image is a synthetic image
However we can still be able to classify this imaginary class
The decision rule for Chestnut sided Warbler is as below. Note that it has a yellow crown.
IF yellow upper-parts is True AND white_nape is True AND primarily white is True AND yellow crown is True
---> THEN predict class Chestnut_sided_Warbler
In this highly interpretable Neural DNF model, we can tweak the rules to classify
Blue-crown Chestnut sided Warbler, even if it does not exist in the training data (or even real world)
(the concept of blue crown, though, is in the training set)
IF yellow upper-parts is True AND white_nape is True AND primarily white is True AND blue crown is True
---> THEN predict class Chestnut_sided_Warbler

Figure 5.8: (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'.

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. 5.8 (top) a correctly predicted sample for the class of 'Chestnut sided Warbler' and in fig. 5.8 (middle) an incorrectly 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 concept which 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. 5.8 (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 expressivity of propositional logic.

In conclusion, our experiments show that Neural DNF achieves accuracy comparable to that of blackbox deep learning models while offering an interpretable symbolic DNF representation that makes it easier for a human to interact with or manipulate the model.

Chapter 6

Conclusion

In this thesis, inspired by the limitations of pure deep learning and pure symbolic models, we investigate the approach of vertical neuro-symbolic integration. We present Neural DNF, a first step of applying vertical integration for the task classification. Neural DNF takes a step towards the fundamental problem of integrating continuous perception and logical reasoning. Neural DNF adopts a two-stage model that utilizes a DNN to extract features, called concept predicates, and a propositional logic DNF module to make classifications based on the concepts. We propose the BOAT algorithm for joint learning of the DNF module and the feature extractor DNN. The introduced adaptive temperatured-noise of BOAT is a minimal yet effective modification that enables the effective optimization of the parameters of the second-stage rule-based model. 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.

As vertical integration essentially consists of a first-stage neural network, a second-stage symbolic model, and the corresponding learning algorithm, below we suggest these three directions for future research respectively.

First-stage neural network. An important direction for future development is to devise novel deep learning architectures with the right inductive bias to extract human-understandable features from data. In this thesis, we mostly focus on the second stage symbolic model and use some generic feedforward neural network with no further constraints or design as the first stage feature extractor. We demonstrate that in some cases the neural network can extract meaningful features that are aligned with human understanding (Sections 5.2 and 5.3) but this is not guaranteed. More importantly, even if the meaning of the feature are aligned with human understanding, it does not mean the computation itself is interpretable. To some degree, in the vertical integration approach, the blackbox problem is only delegated to the feature extractor: if the features extracted by the neural network cannot be understood or verified to align with human understanding, then the blackbox problem will always remain.

We argue that this is a hard problem in two aspects: (1) To obtain interpretable features that identify human-comprehensible concepts, we require very rich and detailed knowledge such as representation learning techniques [Chen et al., 2016] that disentangles concepts, robust training against noise [Khakzar et al., 2019] or extra annotations [Kulkarni et al., 2015, Bau et al., 2017]. And any approach should be highly domain-specific for different data types and tasks [Chen et al., 2019a, Biffi et al., 2018, Kim and Canny, 2017, Johnson et al., 2016]. (2) For features to be *interpretable*, we literally need to properly *interpret* the true meaning of extracted features [Simonyan et al., 2013, Yosinski et al., 2015, Olah et al., 2018, Nguyen et al., 2019]. This is a difficult task and is inevitably vulnerable to confirmation biases, which have been pointed out as a critical issue of interpretability research [Adebayo et al., 2018]. It would be valuable for Neural DNF to adopt previous approaches to develop an interpretable feature extractor, but in general we believe without engineering rich human knowledge this might not be fruitful.

We explored in section 5.5 to use concept annotations to extract interpretable features. As quite an ad-hoc solution, we cannot guarantee the robustness of the feature extractor nor can we understand how the features are extracted. Moreover, in practice often there is no prior knowledgebase of 'concept annotations'. Therefore, novel deep learning architectures that can extract human-understandable concepts from data without the need of concept annotations should be investigated. It is worth noting that such solutions may be highly domain-specific, as it might require domain-specific interpretability criteria and knowledge representations [Melis and Jaakkola, 2018, Chen et al., 2019a, Biffi et al., 2018, Kim and Canny, 2017, Johnson et al., 2016].

Second-stage symbolic model. Another future direction is the use of more powerful languages than propositional logic for specific tasks, for example, more general inductive logic programming. In this thesis, as a natural first choice for classification, we choose propositional logic, formulated in disjunctive normal form. However, the expressivity of propositional logic is certainly limited, even for the classification task. Take MNIST as an example, it is unknown if the true classification task of MNIST can be expressed by propositional logic and neurally-extracted concepts. Perhaps a better solution is to use more powerful languages beyond propositional logic, such as domain-specific spatial grammar that discriminates based on spatial relationships between parts [Lake et al., 2015]. Practically, it will be ideal to have a unified and accessible framework that we can customize a domain-specific symbolic model (programs) for a specific task beyond classification. Closely related and promising work in this direction is the recent development on differentiable program induction [Gaunt et al., 2016, 2017].

The learning algorithm. In this thesis, we introduce the BOAT algorithm for learning Neural DNF. It is a simple algorithm that requires only minimal changes of the standard deep learning optimization workflow, in contrary to a juxtaposed combination of learning algorithms like gradient-based optimization and combinatorial search. We emphasize that it is important for BOAT to be a simple and general algorithm. Technically BOAT is not specific to learning Neural DNF, on the contrary, the algorithm itself does not utilize any knowledge or constraints about DNF. To give a hint on what it means, note that we initialize $\boldsymbol{W}$ and $\boldsymbol{S}$ randomly and let the BOAT algorithm to handle the optimization of $\boldsymbol{W}$ and $\boldsymbol{S}$ based on gradient signals. Conventional combinatorial discrete optimization algorithms are usually aware of the constraints that certain configurations of values of $\boldsymbol{W}$ and $\boldsymbol{S}$ is invalid. For example, in each rule as an AND clause, the indicator of a condition and its negation cannot be set to 1 at the same time, i.e. a rule cannot be 'if $c_0 == 1$ and $c_0 == 0$ then positive' because c_0 is either 0 and 1 and thus this rule is invalid. Discrete optimization algorithms usually need to take this constraint into account, but BOAT does not need such constraints. Empirically we also do not find this to be problematic, even if the initialization configuration turns out to be invalid. This should also be viewed as the advantage of a general and simple learning algorithm. 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.

As for why BOAT works, we mentioned 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 the introduced adaptively-tempratured noise helps learning. We believe that our method BOAT can in principle be linked to approximate variational inference [Khan et al., 2018, Kingma et al., 2015, Potapczynski et al., 2019]. We leave further theoretical investigation of BOAT for future works.

Bibliography

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

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020.

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.

Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: a survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People* *and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6. IEEE, 2017.

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.

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

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.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

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

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`.

Alexander L Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable programs with neural libraries. In *International Conference on Machine Learning*, pages 1213–1222, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

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.

Geoffrey E Hinton. Preface to the special issue on connectionist symbol processing. *Artificial Intelligence*, 46(1-2):1–4, 1990.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

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

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. 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.

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

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.

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.

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.

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.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

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

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

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.

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`.

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.

Marvin L Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI magazine*, 12(2):34–34, 1991.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

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

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

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.

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

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

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.

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

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.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog), March*, 13:12, 2019.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pages 8687–8698, 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.

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.

MIT-IBM Watson AI Lab. Neuro-symbolic ai. `https://mitibmwatsonailab.mit.edu/category/neuro-symbolic-ai/`, 2020. Accessed: 2020-09-30.

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.

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.

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