Title

Efficient and Explainable Machine Learning

Author

Wang, Weijia

Publication Date

2023

UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient and Explainable Machine Learning

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Weijia Wang

Committee in charge:

Professor Bill Lin, Chair
Professor Sanjoy Dasgupta
Professor Sicun Gao
Professor Xiaolong Wang
Professor Pengtao Xie

2023

The Dissertation of Weijia Wang is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to express my heartfelt gratitude to my Ph.D. advisor, Professor Bill Lin, for his unwavering guidance and support during the entirety of my doctoral journey. It was during thought-provoking discussions with him after his lectures that I first found my passion for research, and his mentorship has played an indispensable role in shaping my academic and research development. Without my fortunate encounter with Bill, I might not have chosen to pursue this path. He has not only been my academic advisor but also a cherished friend in my life. I consider myself incredibly fortunate to be pursuing my degree under his expert guidance.

I would like to extend my sincere appreciation to my labmates, Saeed, Litao, and Mahdi, for their invaluable companionship. Their collaborative spirit and support played a crucial role in enhancing my research journey.

I'm also deeply grateful to my Ph.D. committee members, Professor Sanjoy Dasgupta, Professor Sicun Gao, Professor Xiaolong Wang, and Professor Pengtao Xie. Their presence and insights significantly inspired and enriched my work.

Lastly, I would like to convey my wholehearted gratitude to my family, particularly my parents, for their ever-present support and understanding, accompanying me every step of the way.

Chapter 1, in full, is a reprint of the material as it appears in AAAI Conference on Artificial Intelligence, 2021. Litao Qiao*, Weijia Wang*, and Bill Lin. The dissertation author was one of the primary investigators and authors of this paper.

Chapter 2, in full, is a reprint of the material as it appears in IEEE Open Journal of the Computer Society, 2023. Weijia Wang, Litao Qiao, and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in Machine Learning with Applications, 2022. Weijia Wang, Litao Qiao, and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in Mach. Learn. Knowl.

Extr., 2023. Litao Qiao, Weijia Wang, and Bill Lin. The dissertation author was a co-author of this paper.

Chapter 5, in full, is a reprint of the material as it appears in ACM Journal on Emerging Technologies in Computing Systems, 2019. Weijia Wang and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in full, is a reprint of the material as it appears in ACM Journal on Emerging Technologies in Computing Systems, 2022. Weijia Wang and Bill Lin. The dissertation author was the primary investigator and author of this paper.

VITA

2016 Bachelor of Science, Zhejiang University, China

2018 Master of Science, Department of Electrical and Computer Engineering, University of California San Diego

2023 Doctor of Philosophy, Department of Electrical and Computer Engineering, University of California San Diego

FIELDS OF STUDY

Major Field: Computer Engineering

Studies in Machine Learning
Professor Bill Lin

ABSTRACT OF THE DISSERTATION

Efficient and Explainable Machine Learning

by

Weijia Wang

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2023

Professor Bill Lin, Chair

In recent years, neural networks have demonstrated remarkable breakthroughs in perceptual tasks such as computer vision and natural language processing, which achieve exceptional classification accuracy and robust generalization capabilities. My research primarily resides within the domain of deep learning, including two major dimensions:

1. Interpretability: Tabular data plays a crucial role as a primary source of structured information, serving as the foundation of decision-making across various fields, ranging from marketing and healthcare to government policy evaluation. In recognition of its significance, researchers have recently turned their attention to applying neural network models to tabular data due to their generally superior performance in comparison to rule-

based methods. While neural networks excel in performance, they often act as ”black-box” models, posing challenges for applications requiring human interpretability, including medical diagnosis and loan analysis. To address this issue, we propose a series of innovative paradigms aimed at generating human-readable predictions in tabular classification problems through novel neural network training approaches. This allows our interpretable models to leverage the high generalization capacity of neural networks.

2. Efficiency and Scalability: The continuous improvement in neural network performance has come at the cost of increased complexity, including higher storage requirements and computational demands. Despite the substantial processing power of modern hardware for training these models, real-time inference and energy consumption remain significant obstacles, particularly in mobile and wearable applications. To tackle this challenge, we propose to encode deep neural networks using a low-precision number representation, such that the models could achieve accuracy levels comparable to their full-precision counterparts. In addition, we introduce an approach that combines certain steps during the feed-forward phase by pre-computing various intermediate results, allowing the trained neural network to primarily operate in the low-precision domain with fewer floating-point operations.

By addressing these two critical aspects, my work contributes to the advancement of neural network applications in both high-stakes interpretability-sensitive domains and resource-constrained deployment scenarios.

Introduction

During the past few years, the emergence of neural networks has generated considerable excitement across various fields of society. Notably, neural networks have consistently demonstrated significantly superior performance in comparison with conventional machine learning techniques. However, neural networks are generally perceived as "black-box" models due to the challenges for human users seeking to understand their predictions. For instance, even top-tier human players can hardly decipher the rationale behind AlphaZero's [83] moves in a Go game. While such ambiguity might be tolerable in recreational contexts, it significantly hinders neural networks from being applied to the domains involving financial matters and human well-being. On the other hand, traditional interpretable approaches such as rule-based models are usually not as competitive as neural networks and other black boxes in terms of predictive performance. This motivates the first aspect of my research: bridging the gap between the interpretability of decision-rule learners and the effective training techniques of neural networks.

In particular, in Section 1, we first propose the decision rules network (DR-Net), which is a simple three-layer neural network with the neurons carrying out customized arithmetic and the features specifically encoded. The network can be trained with the common gradient-descent-based methods with some regularization techniques, and it after training, directly maps to a decision rule set. Then, in Section 2, we introduce the disjunctive threshold network (DT-Net) which consists of less constrained neurons than DR-Net. Instead of converting the entire network to a rule set, DT-Net itself doesn't directly translate to any specific interpretable model, but the predictions generated by DT-Net come with human-readable explanations. This difference allows DT-Net to achieve higher accuracy in comparison with DR-Net while preserving interpretability.

In addition, we further extend the above approaches with conjunctive threshold network (CT-Net) and Nand-Nand network (NN-Net) as preliminaries of more works in this direction, as discussed in Sections 3, and 4, respectively.

The second facet of my work arises from the growing demands on memory consumption and computation power of neural networks. On one hand, the physical constraints of mobile devices and wearable applications limit their processing capabilities, posing challenges for the deployment of large state-of-the-art network architectures. On the other hand, even for conventional processors, data transfer between the chip and off-chip storage remains challenging. Therefore, we propose to quantize the weights and activations of deep neural networks into low-precision integers that represent certain floating-point numbers with a pair of additional trainable parameters. Our method allows the neural networks to primarily operate with precision levels as low as 2 bits, resulting in negligibly small loss in comparison with the full-precision models.

In Section 5, we explore the utilization of metal-oxide resistive random access memory (ReRAM) for implementing neural network accelerators. In particular, the crossbar structures within ReRAM cells enable the matrix multiplications to be performed in an analog manner. However, the precision of ReRAM-based accelerators is constrained by the resolutions of Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) at the crossbar interface. In this context, our low-precision number representation is proposed to address this limitation. The energy and area estimation of our approach demonstrates a significant reduction compared with the full-precision implementations. Further, in Section 6, we refine our approach and investigate a real-world application where memory constraints pose a significant challenge: 3D image segmentation using 3D U-Nets for brain tumor diagnosis.

Chapter 1

Learning Accurate and Interpretable Decision Rule Sets from Neural Networks

1.1 Introduction

Machine learning is finding its way to impact every sector of our society, including healthcare, information technology, transportation, entertainment, business, and criminal justice. In recent years, machine learning using neural networks have made tremendous advances in solving perceptual tasks like computer vision and natural language processing, with breakthrough performance in classification accuracy and generalization capability. However, neural network methods have generally produced black box models that are difficult or impossible for humans to understand. Their lack of interpretability makes it difficult to gain public trust for their use in high-stakes human-centered applications like medical-diagnosis and criminal justice, where decisions can have serious consequences on human lives [77].

Indeed, interpretability is a well-recognized goal in the machine learning community. One popular approach to interpretable models is the use of decision rule sets [20, 86, 47, 91, 25], where the model comprises an unordered set of independent logical rules in disjunctive normal form (DNF). Decision rule sets are inherently interpretable because the rules are expressed in simple IF-THEN sentences that correspond to logical combinations of input conditions that must be satisfied for a classification. An example of a decision rule set with three clauses is as follows:

IF (age $\leq$ 50) OR
(NOT smoker) OR
(cholesterol $\leq$ 130 AND blood pressure $\leq$ 120)
THEN low heart disease risk.

In this example, the model would predict someone to have a low risk for heart disease if the person's cholesterol level and blood pressure are below the specified thresholds. The model not only provides a prediction, but the corresponding matching rule also provides an explanation that humans can easily understand. In particular, the explanations are stated directly in terms of the input features, which can be categorical (e.g., color equal to red, blue, or green) or numerical (e.g., age $\leq$ 50) attributes, where the binary encoding of categorical and numerical attributes is well-studied [91, 25].

In this paper, we propose a new paradigm for learning accurate and interpretable decision rule sets as a neural network training problem. In particular, we consider the problem of learning an interpretable decision rule set as training a neural network in a simple two-layer fully-connected neural network architecture called a Decision Rules Network (DR-Net). In the first layer, called the Rules Layer, each trainable neuron with binary activation directly maps to a logical IF-THEN rule after training, where a positive input weight corresponds to a positive association of the input feature, a negative input weight corresponds to a negative association of the input feature, and a zero weight corresponds to an exclusion of the input feature. In the second layer, called the OR Layer, the trainable output neuron with binary activation directly maps to a disjunction of the first-layer rules to form the decision rule set.

By formulating the interpretable rules learning problem as a neural net training problem, state-of-the-art training approaches (including recent advances) can be harnessed for learning highly accurate classification models, including well-developed stochastic gradient descent algorithms for effective training. We are also able to leverage well-developed regularization concepts developed in the neural net community to trade off accuracy and model complexity in the training process. In particular, we propose a sparsity-based regularization approach in which the model complexity in terms of the length of the rules and the number of rules are captured in a regularization loss function. Minimizing the number of decision rules makes it easier for a user to understand all the conditions that correspond to a classification, and minimizing the lengths of the decision rules makes it easier for a user to interpret the explanations. This regularization loss function can be combined with a binary cross-entropy loss function that measures training accuracy, so that the training process can balance between classification accuracy and the simplicity of the derived rule set.

Other benefits of a neural net based formulation is the availability of sophisticated development frameworks [1, 68] for model development, powerful computing platforms (e.g., GPUs and deep learning accelerators) for efficient learning and inference, and other developments like federated learning [46] that enables multiple entities to collaboratively learn a common, robust model without sharing data, which addresses critical data privacy and security concerns.

In comparison with previous rule-learning approaches, our approach has several notable advantages. In [47, 91], the pre-mining of frequent rule patterns is first used to produce a set of candidate rules, from which various algorithmic approaches are used to select a set of rules from these candidates. However, the requirement for pre-mining frequent rules limits the overall search space, thus hindering the algorithms from obtaining a globally optimized model. In [86, 25], the problem is formulated as an integer-programming problem in which the pre-mining of rules is not required, but approximations are required to solve large scale problems. In contrast, our neural net based approach does not require rules mining and can take advantage of well-developed neural net training techniques to derive better interpretable models. By connecting interpretable rule-based learning to a neural network based formulation, we hope to open a new line of research that will lead to further fruitful results in the future.

Our experimental results show that our method can generate more accurate decision rule sets than other state-of-the-art rule-learning algorithms with better accuracy-simplicity trade-offs. Further, when compared with uninterpretable black box machine learning approaches such as random forests and full-precision deep neural networks, our approach can easily find interpretable decision rule sets that have comparable predictive performance.

1.2 Related Work

The learning of Boolean rules and rule sets is well studied with different variants. While the learning of two-level Boolean decision rule set has an extensive history in different communities, most of them employ heuristic algorithms that optimize for certain criteria that are not directly related to classification accuracy or model simplicity. Representatives of these methods include logical analysis of data [23, 8], association rule mining and classification [19, 54], and greedy set covering [20].

With the increasing interest in the field of explainable machine learning, researchers have in recent years added model complexity to the optimization objective so that accuracy and simplicity can be jointly optimized. Several approaches select rules from a pre-mined set of candidate rules [91, 47]. A Bayesian framework is presented in [91] for selecting pre-mined rules by approximately constructing a maximum a posteriori (MAP) solution. In [47], the joint optimization problem is approximately solved by a local search algorithm. In these methods, the requirement for rules pre-mining limits the overall search space, hindering their ability to find a globally optimized model. Other approaches based on integer-programming (IP) formulations [86, 25] do not require rules pre-mining, but they rely on approximate solutions for large datasets. In [25], the IP problem is approximately solved by relaxing it into a linear programming problem and applying the column generation algorithm, whereas [86] utilizes various optimization approaches including block coordinate descent and alternating minimization algorithm.

Besides decision rule sets, decision lists [74, 6, 48] and decision trees [9, 75] are also interpretable rule-based models. In decision lists, rules are ordered in an IF-THEN-ELSE sequence. However, the chaining of rules via an IF-THEN-ELSE sequence means that the interpretation of an activated rule requires an understanding of all preceding rules. This can make the explanation more difficult for humans to understand. In decision trees, rules are organized into a tree structure. However, they are often prone to overfitting.

1.3 Decision Rules Network

Given a classification dataset with binarized input features, our goal is to train a classifier in the form of a Boolean logic function in disjunctive normal form (OR-of-ANDs). In particular, each of the lower level conjunctive clauses (logical ANDs), which consists of a subset of input features and their negations, individually serves as a decision rule. An instance satisfies a conjunctive clause if all conditions specified in the clause are true in the instance. In the upper level of the function, all conjunctive clauses are unified by a disjunction (logical OR). Thus, a negative final prediction is produced only if none of the conjunctive clauses are satisfied. Otherwise, a positive final prediction will be made.

Mathematically, the training set contains N data samples $(\mathbf{x}_n, y_n)$, $n = 1, ..., N$, where $\mathbf{x}_n$ comprises D binarized features $x_{n,i} \in \{0,1\}$, $i = 1, ..., D$, and $y_n \in \{0,1\}$. The final decision rule set C learned from our method comprises parallel rules that we denote as clauses: $C = \{c_1, c_2, ..., c_m\}$. We define a clause c to be a conjunction of k predicates where $1 \leq k \leq D$ and a predicate to be either an input feature x_i or the negation of an input feature $\overline{x}_i$. If an input feature or the negation of an input feature is not present in clause c, then we say that feature is excluded from clause c, i.e. whether $x_{n,i}$ is 0 or 1 has no effect to the prediction of clause c. Under this definition, an instance $\mathbf{x}_n$ satisfies a clause only if all predicates in the clause are true in the instance i.e. $x_{n,i} = 1$ for x_i and $x_{n,i} = 0$ for $\overline{x}_i$.

In this section, we introduce the architecture of our Decision Rules Network (DR-Net), which is a simple two-layer fully-connected neural network. The first layer, called the Rules Layer, consists of trainable neurons that map to logical IF-THEN rules, and the second layer, called the OR Layer, contains a trainable output neuron that maps to a disjunction of the first-layer rules to form the decision rule set. The goal of the design of this network is to simulate the logical formula in disjunctive normal form so that a trained DR-Net can be directly mapped to a set of interpretable decision rules.

1.3.1 Handling of Categorical and Numerical Attributes

Common tabular datasets generally comprise binary, categorical and numerical features. While our method is based on binary encoded input vectors, we employ the following pre-processing procedures, which are well established and studied in the machine learning literature, to binarize the input features. In particular, the values of binary features are left as what they are, whereas we apply standard one-hot encoding to transform categorical attributes to vectors of binary values. As for numerical features, we adopt quantile discretization to get a set of thresholds for each feature, where the original numerical value is one-hot-encoded into a binary vector by comparing with the thresholds (e.g., age $\leq$ 25, age $\leq$ 50, age $\leq$ 75) and encoded as 1 if less than the threshold or 0 otherwise. For example, considering a dataset that consists of the categorical feature "color" chosen from $\{$red, green, blue$\}$ and a numerical feature "age" with thresholds $\{25, 50, 75\}$, our pre-processing approach will encode an instance [color: red, age: 30] as [red, green, blue, age $\leq$ 25, age $\leq$ 50, age $\leq$ 75] $=$ [1, 0, 0, 0, 1, 1]. Most other rule-learning methods [91, 25] require to convert binary, categorical and numerical features into both positive conditions, e.g., (color = blue) and (age $\leq$ 50), and negative conditions, e.g., (color $\neq$ blue) and (age $>$ 50), of the binary vectors in their pre-processing procedures. On the other hand, our encoding approach only involves those positive conditions without separately having their negations included. Further explanations will be discussed in the next section.

1.3.2 Rules Layer

The essence of a fully-connected layer is the dot-product operation shifted by a bias term. In this context, we notice that with binarized input features, a neuron can be constructed such that it effectively performs a logical AND operation by dynamically adjusting the bias based on the weight values and applying a binary step activation function afterwards. Then, by interpreting the full precision weights in a certain way, each neuron is effectively a conjunction of input features and thus the whole layer can be mapped to a set of clauses that can be later combined with disjunction to form a DNF rule set.

Mathematically, given the input to the Rules Layer as $\mathbf{x} \in \{0,1\}^D$ and the output as y, a neuron in the Rules Layer performs its operation as follows:

$$y = \sum_{i=0}^{D} w_i x_i - \sum_{w_i > 0} w_i + 1. \tag{1.1}$$

In Equation 1.1, the dot product of the weights and inputs is added with a dynamic bias, which depends on the weights of the neuron. With the dynamic bias and binarized inputs, the range of the outputs of the neurons in the Rules Layer is within $(-\infty, 1]$. Note that the output $y = 1$ can only be achieved when all inputs match the sign of the corresponding weights: all positive weights should have the inputs of 1 and all negative weights should have the inputs of 0. Just like the behavior of weights in regular neurons, the zero weights in the Rules Layer mean that the corresponding inputs will not have any effect on the output.

In order for the neuron in the Rules Layer to function as a proper logical AND operation, we need to apply a binary step activation function to its output:

$$f(x) = \begin{cases} 1 & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

When applied at the Rules Layer, the binary step function defined in Equation 1.2 simply maps the range $(-\infty, 1)$ to 0, which ensures that the neuron is turned on only when Equation 1.1 evaluates to 1. With the dynamic bias and binary step function, each neuron in the Rules Layer encodes a rule that has k predicates, where k is the number of non-zero weights of that neuron. As discussed earlier, in effect neuron in the Rules Layer maps to a logical IF-THEN rule after training, where a positive input weight corresponds to a positive association of the input feature, a negative input weight corresponds to a negative association of the input feature, and a zero weight corresponds to an exclusion of the input feature.

However, as can be observed, the activations of the first layer are discretized into binary integers that are not naturally differentiable and the classic gradient computation approach doesn't apply here. Therefore, we utilize the straight-through estimator discussed in [5] with the gradient clipping technique. Denoted by $\hat{y}_i$ the binarized activation based on y_i, we compute the gradient as follows:

$$g_{\hat{y}_i} = \begin{cases} 0 & \text{if } y_i < 0 \\ & \text{or } y_i > 1 \frac{\partial L}{\partial y_i} < 0 \\ g_{y_i} & \text{otherwise} \end{cases} \tag{1.3}$$

where $g_{\hat{y}_i}$ and g_{y_i} are the gradients of classification loss w.r.t. $\hat{y}_i$ and y_i, respectively. The condition $y_i < 0$ simulates the backward computation of the ReLU function, which introduces non-linearity into the training process and empirically improves the performance; whereas our motivation of the second condition is to address the saturation effect: we suppress the update of the full-precision activations that are greater than 1 and are still driven by the gradient to increase, since further raising activations does not produce any difference after binarization.

As discussed in above, the addition of the negative conditions in the input space is critical to the selection-based methods [91, 25] since they only consider the presence and absence of features and cannot deduce negative correlations unless they are explicitly provided in the input space. On the other hand, besides the presence of a positive association or an exclusion, our Rules Layer also learns the negation of an input feature by assigning a negative weight to it, and hence, DR-Net can directly derive negative conditions from the corresponding input features. Therefore, appending negative conditions in the input binary vector is redundant in DR-Net, and the input space of our DR-Net is reduced by half comparing with those selection-based method.

1.3.3 OR Layer

To produce the disjunction of the logical rules learned in the Rules Layer, the OR Layer contains only one output neuron, where the weights are binarized as follows:

$$\hat{w}_i = \begin{cases} 0 & \text{if } w_i \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{1.4}$$

The output neuron performs a dot product with a negative bias $-\varepsilon$ as follows:

$$y = \sum_{i=1}^{D} \hat{w}_i x_i - \varepsilon, \tag{1.5}$$

where $0 < \varepsilon < 1$ is a small value such that y is positive when at least one input is activated. With a sigmoid activation function and a binary cross-entropy loss, this particular neuron behaves as an OR gate: the output is by default turned off because of the negative bias, while it produces a positive value if at least one rule is activated with a corresponding $\hat{w}_i = 1$, which exactly mimics the behavior of the logical OR function. The binarized weights $\hat{w}_i$ act as rule selectors that filter out rules that do not contribute to the model's predictive performance. An example of our complete network structure is shown in Figure 1.1. We practically use $\varepsilon = 0.5$ in our implementations.

1.3.4 Sparsity-Based Regularization

The neural network structure proposed above outlines a way to derive a set of decision rules using stochastic gradient descent. As discussed above, a zero weight for a Rules Layer neuron corresponds to the exclusion of the corresponding input feature. Similarly, a zero weight for the OR Layer output neuron corresponds to the exclusion of the corresponding rule from the rule set. Thus, it should be clear that maximizing the *sparsity* of the Rules Layer neurons corresponds to simplifying the corresponding rules, and maximizing the sparsity of the OR Layer neuron corresponds to minimizing the number of rules.

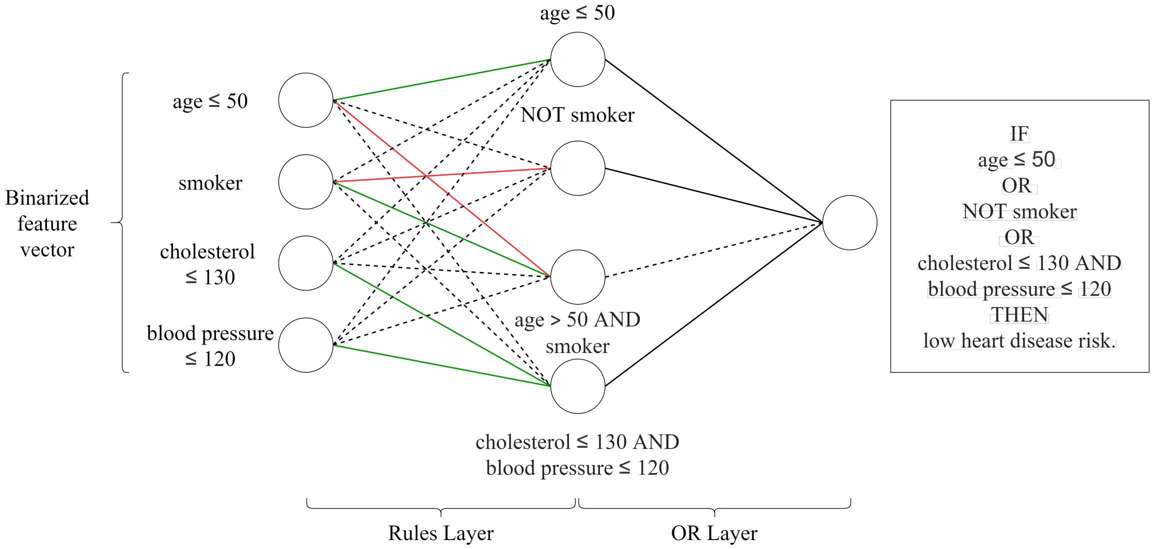


Figure 1.1. An example of the DR-Net architecture where three rules from the Rules Layer are included in the OR Layer, and one rule is excluded. The decision rule set that the network directly maps into is shown in the box on the right. The dashed lines represent the masked weights (weights that are set to zero). The green lines in the Rules Layer represent positive weights while red lines represent negative weights. Please note that we represent (NOT age ≤ 50) as (age > 50) in the third rule, and it is not included in the final rule set because it has been masked in the OR Layer.

However, to eliminate an input feature from a logical rule or a logical rule from the complete rule set, the corresponding weight has to be exactly zero, which is difficult to achieve in the typical network training process. To achieve a high degree of sparsity with exact zero weights, we explicitly incorporate a *sparsity-based regularization* mechanism into the training process using an approach akin to L_0 regularization by explicitly training *mask* variables.

As discussed in [58] as a way to achieve network sparsity through L_0 regularization, a binary random variable $z_i \in \{0,1\}$ is attached to each weight of the model to indicate whether the corresponding weight is kept or removed. With this, we can reparameterize each weight w_i as the product of a weight $\tilde{w}_i$ and the corresponding binary random variable z_i:

$$w_i = \tilde{w}_i z_i. \tag{1.6}$$

Assuming each z_i is subject to a Bernoulli distribution with parameter π_i, i.e. $q(z_i|\pi_i) = \text{Bern}(\pi_i)$, the probability that z_i is 1 is just π_i. In [58], L_0 regularization is implemented by summing all π_i parameters as the penalty term in the loss function[1]. In order to train the binary random variables with stochastic gradient descent, two different gradient estimators have been proposed in [58] and [51], respectively, to approximate the Bernoulli distribution.

Applying the above regularization method to the Rules Layer is straightforward: all weight parameters are replaced by their product with the corresponding mask variables. For the OR Layer, since the weights will ultimately be binarized, we can just directly substitute the mask variables for the weights to simplify the process. That is, we can simply treat $w_i = z_i$, with no need for a separate $\tilde{w}_i$ variable[2].

We then incorporate a sparsity-based regularization term in the loss function to model the complexity of the rule set represented by the neural network. We denote by $\pi_{1,i,j}$ and $\pi_{2,j}$ the penalty of the non-zero mask variables of the Rules Layer and the OR Layer, respectively, where $i = 1, 2, \ldots, D$ is the feature index, and $j = 1, 2, \ldots, m$ is the index to the j-th neuron (rule). Then the regularization loss is defined as follows:

$$\mathscr{L}_R = \frac{1}{m}\left(\sum_{j=1}^{m} \pi_{2,j} + \sum_{j=1}^{m} \pi_{2,j} \sum_{i=1}^{D} \pi_{1,i,j}\right), \tag{1.7}$$

which *explicitly* captures the model complexity, as similarly defined in [25]. In particular, the model complexity of a rule set is defined as the sum of the number of rules and the total number of predicates in all rules. Following this definition, the first and the second terms of Equation 1.7 quantify the losses for the number of rules and the total number of predicates, respectively. Note that, according to the second term in Equation 1.7, the loss for the number of predicates in the j-th rule will be effectively removed if $\pi_{2,j}$ is at or near zero: i.e., the j-th neuron in the Rules

[1] As explained below, we do not use L_0 normalization as the regularization term. Instead, we explicitly capture the model complexity with Equation 1.7 as the regularization term.

[2] Since $w_i = z_i$ is already binarized, there is no need to further binarize w_i to derive $\hat{w}_i$ with Equation 1.4. i.e., we can just use $\hat{w}_i = w_i = z_i$.

Layer is disconnected from the OR Layer.

With the above sparsity-based regularization applied to DR-Net, the overall loss function we optimize for can be expressed as follows:

$$\mathscr{L} = \mathscr{L}_{\text{BCE}} + \lambda \mathscr{L}_R, \tag{1.8}$$

where $\mathscr{L}_{\text{BCE}}$ is the binary cross-entropy loss, $\mathscr{L}_R$ is the regularization penalty that is specified by Equation 1.7, and λ is the regularization coefficient that balances the classification accuracy and rule set complexity.

1.3.5 Alternating Two-Phase Training Strategy

As previously discussed, each neuron in the first layer (the Rules Layer) of our proposed network architecture encodes an interpretable decision rule, whereas the output neuron in the second layer (the OR Layer) chooses some of the rules to be included in the set of decision rules. Empirically, we noticed that it is more effective to train our DR-Net with gradient-based optimizers (e.g., SGD) in an alternating manner, potentially due to the reduced search space and simpler optimization goals. In particular, our "alternating training strategy" consists of two training phases. We first freeze the OR Layer and only update the parameters in the Rules Layer to learn plausible rules. In the second phase, the Rules Layer is then fixed and we optimize the OR Layer such that redundant rules are eliminated while necessary inactive rules can also be re-enabled. The whole network is trained by alternating between the two training phases until convergence.

In addition, since the sparsity of the Rules Layer is directly related to the simplicity of rules, whereas the second layer is more focused on the selection of these derived rules, we further allow the flexible weighting of the sparsity-based regularization loss of the two layers. Specifically, as illustrated in Equation 1.8, the balance between classification loss and regularization loss is implemented via the regularization coefficient λ, where we can practically use different values for the two phases. In other words, the Rules Layer and the OR Layer are optimized over $\mathscr{L}_1$ and $\mathscr{L}_2$, respectively, where $\mathscr{L}_1$ and $\mathscr{L}_2$ are defined as follows:

$$\begin{aligned} \mathscr{L}_1 &= \mathscr{L}_{\text{BCE}} + \lambda_1 \mathscr{L}_R, \\ \mathscr{L}_2 &= \mathscr{L}_{\text{BCE}} + \lambda_2 \mathscr{L}_R. \end{aligned} \tag{1.9}$$

In this way, the trade-off between model simplicity and accuracy in our experiments can be modulated by the adjustments of λ_1 and λ_2.

1.4 Experimental Evaluation

The numerical experiments were evaluated on 4 publicly available binary classification datasets, which all have more than 10,000 instances and more than 10 attributes for each instance before binarization. The first two selected datasets are from UCI Machine Learning Repository [27]: MAGIC gamma telescope (magic) and adult census (adult), which are also used in recent works on rule set classifiers [24, 91, 25]. The magic dataset is a dataset with pure numerical attributes while the adult dataset has a mix of both categorical and numerical attributes. The other two datasets are relatively recent datasets: the FICO HELOC dataset (heloc) and the home price prediction dataset (house), which have all numerical attributes. In all datasets, pre-processing is performed to encode categorical and numerical attributes into binary variables, as discussed earlier in the paper. Also, we append negative conditions for all other models except DR-Net.

Our goal is to learn a set of decision rules using our DR-Net and compare our model with other state-of-the-art rule learners and machine learning models. The results include model accuracies and complexities. Apart from the model complexity defined earlier (the number of rules plus the total number of conditions in the rule set), we also define the rule complexity, which is the average number of conditions in each rule of the model. We consider three other rule learners to directly compare with our work in terms of both accuracy and interpretability: the RIPPER algorithm [20], Bayesian Rule Sets (BRS) [91], and the Column Generation (CG) algorithm from [25]. The first one is an old rule set learning algorithm that is a variant of the Sequential Covering algorithm, while the other two are representatives of recent works in rule learning classifiers. We used open-source implementations on GitHub for all three algorithms, where the CG implementation [3] is slightly modified from the original paper. Other models used for comparison are the scikit-learn [69] implementations of the decision tree learner CART [9] and Random Forests (RF) [10]. We also include a full-precision deep neural network (DNN) model with 6 layers, 50 neurons per hidden layer and ReLU activations. The last two models are *uninterpretable* models intended to provide baselines for typical performances that black-box models can achieve on these datasets. These uninterpretable baseline results serve as benchmarks for accuracy comparisons.

For DR-Net, we used the Adam optimizer with a fixed learning rate of 10^{-2} and no weight decay across all experiments. There are 50 neurons in the Rules layer to ensure there is an efficient search space for all datasets. The alternating two-phase training strategy discussed earlier is employed with 10,000 total number of training epochs and 1,000 epochs for each layer. For simplicity, the batch size is fixed at 2,000 and the weights are uniformly initialized within the range between 0 and 1. The parameters that are related to sparsity-based regularization are set the same as in the original paper [58].

1.4.1 Classification Performance

We evaluated the predictive performance of DR-Net by comparing both test accuracy and complexity with other state-of-the-art machine learning models. 5-fold nested cross validation was employed to select the parameters for all rule learners that explicitly trade-off between accuracy and interpretability to maximize the training set accuracies. To ensure that the final rule learner models are interpretable, we constrained the possible parameters for nested cross validation to a range that results in a low model complexity. For DR-Net, We fixed the λ_2 to be 10^{-5} in Equation 1.9 and only λ_1 was varied in the experiment. Although there are many parameters in BRS to control the rule complexity, we followed the procedure used in [25] and only varied the multiplier κ in prior hyper-parameter to save running time. For RIPPER, we varied the maximum number of conditions and the maximum number of rules as hyper-parameters of the implementation, which are directly related to the complexity of the model. The CG implementation in [3] doesn't have the complexity bound parameter C as specified in [25] but instead provides two hyper-parameters to specify the costs of each clause and of each condition, which were used in our experiment to control the rule set complexity. We left all other parameters for these three algorithms (CG, BRS, RIPPER) as default. For CART and RF, we constrained the maximum depth of trees to be 100 for all datasets to achieve better generalization. For DNN, we used the same training parameters (number of epochs, batch size, learning rate, etc.) with a weight decay of 10^{-2}. The test accuracy results of all models on all datasets are shown in Table 1.1 and the corresponding complexities are shown in Table 1.2. We omitted the results of the complexities of CART, RF and DNN because they have a different notion of model complexity and rule complexity.

It can be seen in Table 1.1 that our method outperforms other interpretable models on all datasets. For these better accuracy results, our method does not establish a similar superiority in the complexity comparison (Table 1.2). However, as shown Figure 1.2 and further discussed in the next section, our DR-Net approach can often achieve higher accuracy at comparable complexities. It is interesting to see that DR-Net maintains a relatively good model complexity compared with the corresponding rule complexity, which is exactly because our regularization loss function is designed specifically to minimize the model complexity instead of the rule complexity. Compared with RIPPER, which greedily mines good rules in each iteration to maximize the training accuracy, DR-Net is very competitive in the sense that it has similar or better test performance while consistently maintaining a lower model complexity. One advantage of the BRS algorithm over other models is that it consistently generates sparse models across all datasets, but at the expense of significantly inferior accuracies. The CART decision tree algorithm turned out to be the worst performing model in our experiments, which might result from overfitting. The results in Table 1.1 and Table 1.2 suggest that our DR-Net approach is very

Table 1.1. Test accuracy based on the nested 5-fold cross validation (%, standard error in parentheses).

dataset	magic	adult	heloc	house
interpretable				
DR-Net	**84.42**	**82.97**	**69.71**	**85.71**
	(0.53)	(0.51)	(1.05)	(0.40)
CG	83.68	82.67	68.65	83.90
	(0.87)	(0.48)	(3.48)	(0.18)
BRS	81.44	79.35	69.42	83.04
	(0.61)	(1.78)	(3.72)	(0.11)
RIPPER	82.22	81.67	69.67	82.47
	(0.51)	(1.05)	(2.09)	(1.84)
CART	80.56	78.87	60.61	82.37
	(0.86)	(0.12)	(2.83)	(0.29)
uninterpretable				
RF	86.47	82.64	70.30	88.70
	(0.54)	(0.49)	(3.70)	(0.28)
DNN	87.07	84.33	70.64	88.84
	(0.71)	(0.42)	(3.37)	(0.26)

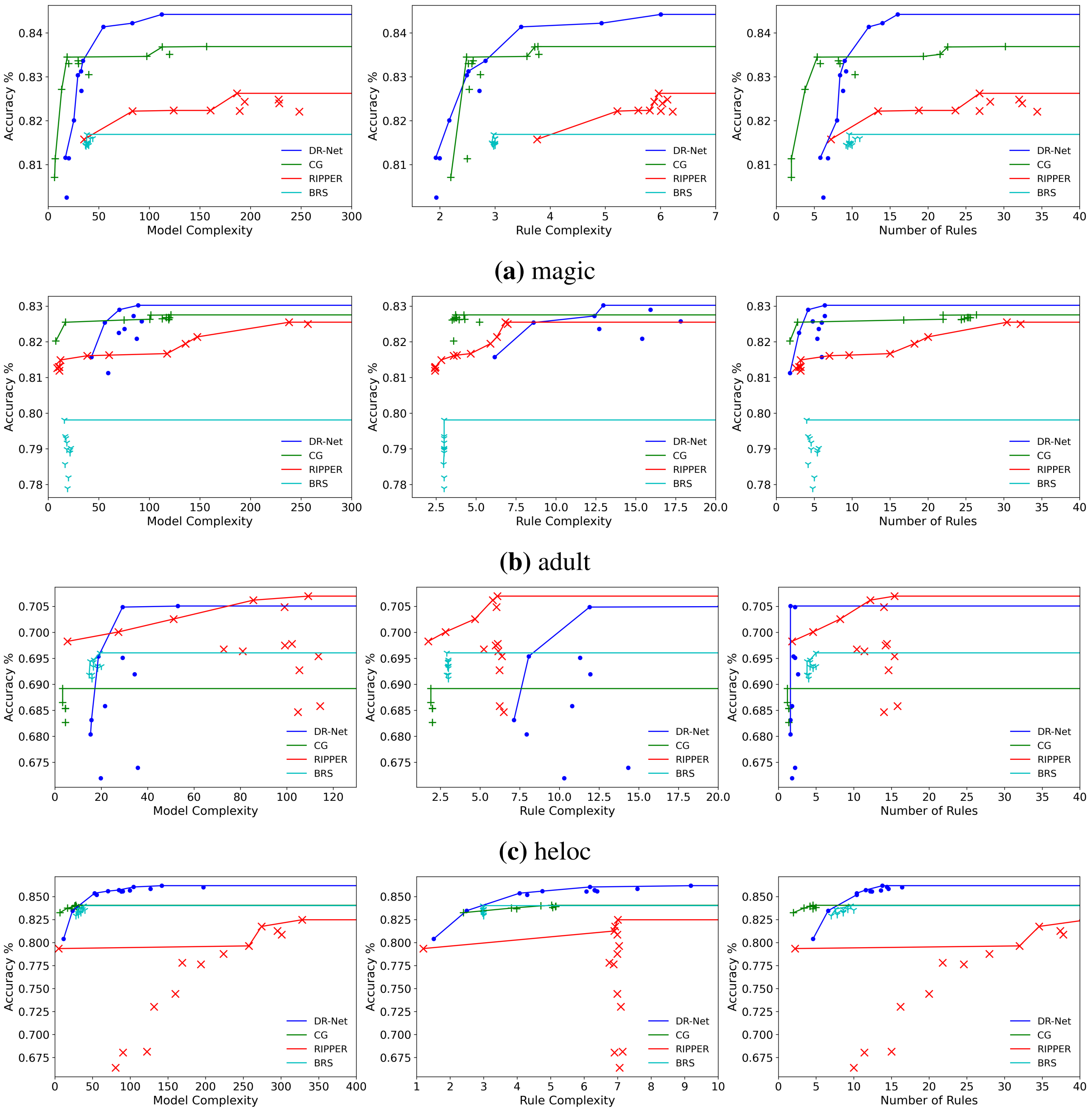
competitive as a machine learning model for interpretable classification. Finally, our DR-Net approach is able to achieve accuracies within only 3% of the uninterpretable models (RF and DNN) on the datasets evaluated.

Table 1.2. Model complexity (upper) and rule complexity (lower) corresponding to the accuracy results shown in Table 1.1 based on the nested 5-fold cross validation. While DR-Net, using parameters selected by the nested 5-fold cross validation with the priority for accuracy, does not achieve the best complexity in comparison with other models, it can be observed in Figure 1.2 that our approach can generally achieve a higher accuracy at the cost of comparable complexities.

dataset	magic	adult	heloc	house
DR-Net	109.4	86.0	13.8	85.0
	5.22	13.54	6.33	6.31
CG	112.8	120.0	**3.4**	**28.6**
	3.72	3.77	**1.90**	5.15
BRS	**40.0**	**16.8**	16.6	31.2
	3.00	**3.00**	2.96	**3.00**
RIPPER	189.4	117.6	72.8	328.0
	6.01	4.66	5.24	7.01

1.4.2 Accuracy-Complexity Trade-off

In this experiment, we compared the accuracy-complexity trade-off of our DR-Net with other rule learning algorithms: CG, BRS and RIPPER. The parameters that were selected to be varied in this experiment are the same as the ones in the first experiment. Instead of using nested cross validation to select best parameters on the validation set, we manually picked a set of values for each selected parameters for each algorithm to generate different sets of accuracy-complexity pairs. We ran the experiments on all datasets and the results with the average of the 5-fold cross validation are shown in Figure 1.2. Apart from model complexity and rule complexity, we included a third metric to show the average number of rules in each generated rule set versus the test accuracy. For each method compared, the dots connected by the line segments shown correspond to Pareto efficient models where all other points below the Pareto frontier have either



(a) magic

(b) adult

(c) heloc

(d) house

Figure 1.2. Accuracy-Complexity trade-offs on all datasets. Pareto efficient points are connected by line segments.

lower accuracies or higher complexities.

The characteristic of being able to attain a high test accuracy with an acceptable model complexity for DR-Net in Table 1.1 and Table 1.2 is carried over to Figure 1.2. For the magic, adult and house datasets, DR-Net outperforms all other rule learners in terms of the accuracy by a substantial margin when the model complexity, the rule complexity or the number of rules exceeds a certain threshold. Although DR-Net does not dominate RIPPER on the heloc dataset, their accuracy comparison is very close if enough model complexity or number of rules is given. The only thing that DR-Net falls behind a little bit is in the rule complexity vs. accuracy comparison on the heloc dataset. In theory, DR-Net can achieve relatively low rule complexity with a different regularization loss function that can quantify the average number of conditions in the rule set, which we leave as future work. It is also interesting to note that the number of rules from DR-Net varies in a relatively narrower range compared with other approaches as shown in the third column of Figure 1.2, which is directly resulted by fixing λ_2 in Equation 1.9. BRS does not demonstrate a clear accuracy-complexity trade-off as its results all group in a very narrow range, which is also noted and explained in [25]. This experiment shows that DR-Net can be preferred over other rule learners because of its potential for achieving a much higher test accuracy with a relatively moderate complexity sacrifice.

1.5 Conclusion and Extensions

In this paper, we presented a simple two-layer neural network architecture, which can be directly mapped to a set of interpretable decision rules, along with a procedure to accurately train the network for classification. We described a sparsity-based regularization approach that can capture the complexity of the trained model in terms of the length of the rules and the number of rules. The incorporation of this regularization loss into the overall loss function enables the training process to balance between classification accuracy and model complexity. With our neural net formulation, we are able to leverage state-of-the-art neural net infrastructures to learn highly accurate and interpretable rule-based models. Our experimental results show that our method can generate more accurate decision rule sets than other state-of-the-art rule-learners with better accuracy-simplicity trade-offs. When compared with uninterpretable black box models such as random forests and full-precision deep neural networks, our approach can easily learn interpretable models that have comparable predictive performance.

We focus in this paper on the binary classification problem, but the approach can be easily extended to multi-class classification by deploying separate output neurons for each class and mapping each output neuron to a corresponding set of rules for the respective class. A default class and a tie-breaking function could be used in the event that no class or more than one class is activated, respectively [47], or these cases can be handled by error correcting output codes [78]. We plan to investigate in future work potentially more powerful tie-breaking mechanisms that can be directly trained as part of the neural net formulation, for example by directly interpreting softmax results.

Acknowledgements

This research has been supported in part by the National Science Foundation (NSF IIS-1956339).

1.6 Acknowledgements

Chapter 1, in full, is a reprint of the material as it appears in AAAI Conference on Artificial Intelligence, 2021. Litao Qiao*, Weijia Wang*, and Bill Lin. The dissertation author was one of the primary investigators and authors of this paper.

Chapter 2

Disjunctive Threshold Networks for Tabular Data Classification

2.1 Introduction

Machine learning is finding its way to impact every sector of our society, including healthcare, transportation, finance, retail, and criminal justice. In high-stakes human-centered applications like medical-diagnosis and criminal justice, where decisions can have serious consequences on human lives, the critical importance of interpretability to explain predictions or decisions is well-recognized in the machine learning community [77].

2.1.1 Related Work

One popular approach to interpretable models is the use of decision rule sets [20, 47, 91, 25], which are inherently interpretable because the rules are expressed in simple if-then sentences that correspond to logical combinations of input conditions that must be satisfied for a classification. Besides decision rule sets, decision lists [74, 48] and decision trees [9] are also interpretable rule-based models. Not only do these decision models provide predictions, but the corresponding matching rules also serve as human-understandable explanations.

Gradient boosting decision trees [15, 44] and random forests [10] have also been successfully used in learning problems involving tabular data. Although these methods provide superior predictive performance in comparison with design rule learning, they are generally considered to be lacking in interpretability, which may limit their adoption in certain application domains.

Neural networks have also been recently proposed for tabular data classification [53, 43, 2]. The work in [43, 2] introduces additional inductive bias to over-parameterized neural networks by designing specific neural network structures to emulate the axis-aligned splits of decision trees that have made the ensembles of trees so successful for tabular datasets. Although both works leverage feature selection techniques as part of their structure design, which can be extracted to interpret the feature attributions to the prediction or classification, this level of interpretability is very limited compared to rule-based sentences that can be easily understood by humans.

In contrast, the recent work in [53] proposed a neural network model that is specifically designed have an underlying disjunctive normal form representation of a decision rule set. To achieve this one-to-one correspondence, the hidden layer neurons in the proposed model are restricted in a manner so that they directly map to conjunctions (logical-ANDs) of input features. These conjunctions correspond to interpretable decision rules. The output neuron implements a disjunctive (logical-OR) operation that aggregates the interpretable decision rules in the hidden layer into a decision rule set. The proposed solution has the same advantage as the class of decision rule learning and tree approaches [20, 47, 91, 25, 9] in that it can also provide meaningful explanations, but it is able to do so with superior predictive performance. However, the approach in [53] imposes restrictions on the hidden layer neurons in a way that limits the search space.

There is also a body of work [82, 81, 18, 36, 40, 4] on compiling models into tractable forms. The tractable form can then be analyzed to produce explanations. In contrast, our approach derives human understandable explanations directly from our proposed model using a fast and simple algorithm.

2.1.2 Our Contribution

We propose to address the tabular data classification problem with a new neural network model called DT-Net (Disjunctive Threshold Network). The hidden layer neurons in the proposed model are trained with floating point weights and binary output activations. These neurons can be interpreted as threshold logic functions, which provides considerably greater flexibility than the DR-net [53] approach that restricts hidden layer neurons to implement conjunction (AND) operations. State-of-the-art techniques can be used to train the proposed neural network model to achieve high predictive performance. Unlike traditional black-box approaches like gradient boosting trees, random forests, and conventional neural networks, DT-Net can also provide rule-like explanations that are comprehensible to humans. However, unlike prior work on decision rule learning [91, 25, 53], our approach does not require the explicit construction of a decision rule set. This means that our disjunctive network of threshold functions can implicitly encode a potentially complicated set of rules to achieve high predictive performance, but yet the derived explanations can nonetheless be simple.

The remainder of the paper is organized as follows: Section 2.2 describes our proposed DT-Net architecture. Section 2.3 describes how explanations can be efficiently derived from a DT-Net inference. Section 2.4 describes how sparsity-inducing regularization can help to simplify explanations. Our proposed approach is extensively evaluated in Section 2.5. Finally, concluding remarks are given in Section 2.6.

2.2 Disjunctive Threshold Network

We introduce in this section the Disjunctive Threshold Neural Network architecture, or DT-Net for short. It is aimed at tabular classification problems in which the ability to *explain decisions* is essential, in addition to making accurate predictions. DT-Net is a simple three-layer neural network architecture comprising n input units, k hidden layer units, and a single output unit. A toy example of the proposed architecture is shown in Figure 2.1, which we use to explain the main points of our work.

Input layer: Each of the n units at the input layer passes its corresponding assigned binarized value to each neuron in the hidden layer. Generally, tabular datasets can have input attributes

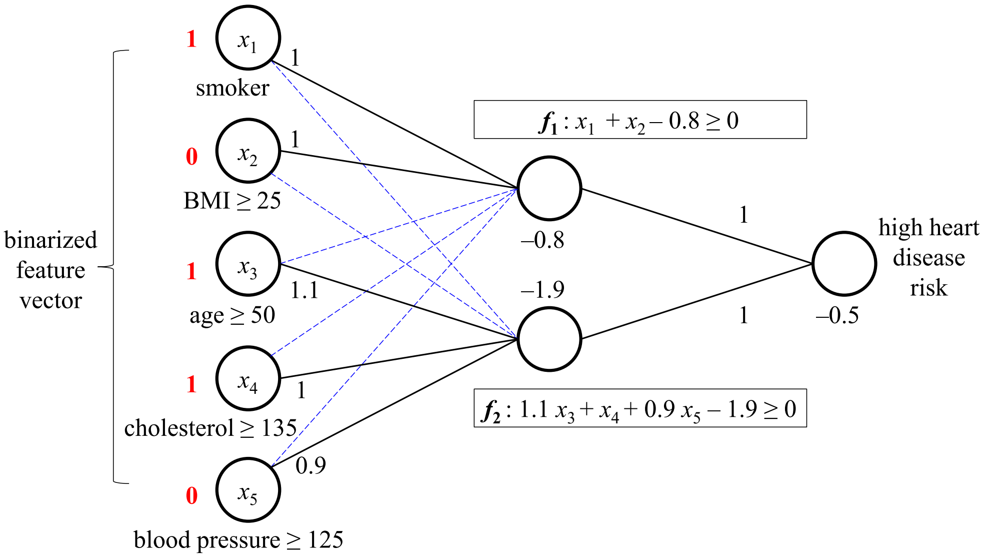


Figure 2.1. An example of the DT-Net architecture. Each hidden layer unit implements a threshold logic function, and the output unit implements a disjunction of these threshold functions. Explanations can be readily derived from the network to explain positive predictions.

that are binary, categorical, or numerical. To handle categorical and numerical attributes, well established and studied pre-processing procedures in the machine learning literature can be used to encode them into binarized input vectors. In particular, standard one-hot encoding can be used to transform categorical attributes into binary vectors, and standard quantile discretization can be used to encode numerical values into binary vectors[1].

Hidden layer of threshold functions: Each of the *k* units in the hidden layer is a threshold function that is trainable with arbitrary (positive or negative) full-precision weights and biases. This is implemented using a binary step activation function. The blue dashed lines in Figure 2.1 indicate that the corresponding features have zero weights, which means the corresponding threshold function is not dependent on them. As discussed in the next section, each threshold function implicitly encodes an underlying Boolean logic function of inputs that will yield to a positive result.

Output disjunction layer: The output layer is designed to implement a *disjunction* of the *k*

[1] Widely studied interpretable rule-learning methods [91, 25, 53] on tabular classification problems also commonly assume the datasets to be binarized through pre-processing.

hidden layer threshold functions, which consists of a single neuron with all weights and the bias fixed at 1 and $-\varepsilon$, respectively, where ε is a small constant between 0 and 1 (we use $\varepsilon = 0.5$ in our experiments). This output threshold unit implements a logical-OR operation since by default, it makes a negative prediction if none of the threshold functions in the hidden layer is activated, whereas any activated threshold function is sufficient to cause the output unit to make a positive prediction. Since each threshold function essentially encodes an underlying Boolean logic function, the whole network also implicitly implements a Boolean logic function by taking the disjunction of these threshold functions. We note that these two layers together compose a logic function in disjunctive normal form, which is capable of encoding any possible Boolean logic function. In other words, our proposed model is applicable to any binary classification problem.

Straight-through estimator: As previously mentioned, the outputs of the threshold functions are fed into a step activation function, which has an impulse derivative function that prevents the gradients from propagating through. In this work, we adopt the straight-through estimator with the gradient clipping technique to address this issue, which is detailed as follows:

$$g_{\hat{z}_i} = \begin{cases} 0, & \text{if } z_i < 0 \text{ or } (z_i > 1, g_{z_i} < 0) \\ g_{z_i}, & \text{otherwise} \end{cases} \tag{2.1}$$

where $g_{\hat{z}_i} = \frac{\partial L}{\partial \hat{z}_i}$ and $g_{z_i} = \frac{\partial L}{\partial z_i}$ are respectively the gradients of classification loss with respect to $\hat{z}_i$ and z_i.

Similar to the ReLU activation function, the step function only produces non-negative outputs. Therefore, we follow ReLU and clip the gradient w.r.t. negative outputs. Moreover, since the step function has an upper bound of 1 for its output, further increasing an activation that is already greater than 1 does not make any improvement, which empirically can even lead to an explosion of the weights. Therefore, we propose to clip such gradient that tries to further increase an activation greater than 1.

Example: Consider the heart disease risk prediction example again, as depicted in Figure 1. Each input instance corresponds to an individual and the features of this person, i.e., smoker, overweight, older than 50, cholesterol, and high blood pressure, are encoded as $x_1, x_2, \ldots, x_5$, respectively. In this toy example, threshold function (hidden neuron) f_1 can be activated by the individual being either a smoker or overweight, and threshold function f_2 evaluates to true if at least two out of the three features with non-zero weights (older than 50, high cholesterol, and high blood pressure) are 1, due to the fact that for any combination of at least two of these features, the summation of their weights is sufficiently greater than 1.9. Given the individual represented as $\langle x_1, x_2, x_3, x_4, x_5 \rangle = [10110]$, both neurons f_1 and f_2 produce a 1 for this instance. Therefore, the entire network produces a positive prediction (the individual has a high heart disease risk).

There can be several *explanations* as to *why* the individual is predicted to have a high heart disease risk. One explanation is that the individual is a *smoker*, which sufficiently explains the high heart disease risk prediction. This explanation is also the *simplest explanation* in that there is no other explanation that is more concise. A more complex explanation is that the person is *older than 50* with *high cholesterol*. This explanation is the simplest when only considering f_2, but it is not the simplest explanation overall as identifying the individual as a smoker is a more concise explanation. However, it is a *minimal explanation* in that no other condition can be removed from the explanation so that the explanation remains sufficient: i.e., *older than 50* by itself is insufficient to explain a high heart disease risk prediction. As detailed later in the paper, given a positive prediction, we can easily *derive* the simplest explanation with respect to an activated threshold function.

Unlike existing interpretable rule-learning methods [91, 25, 53] that explicitly generate sets of decision rules as classifiers, our approach does not require the generation of any specific decision rule set from the trained disjunctive threshold network model. Instead, predictions are made through standard neural network operations so that potentially complicated rules can still be implicitly encoded to achieve better generalizations, where simple explanations for each positive prediction can nonetheless be readily generated afterwards. In addition, due to the natural use of stochastic gradient descent (SGD), any state-of-the-art SGD training techniques can be applied to improve the classification performance. In particular, we will discuss later in the paper a well-developed sparsity-inducing method that we incorporate to simplify the network, which further leads to concise explanations. In the next section, we describe how human-readable explanations can be readily derived for positive predictions produced by the proposed network.

2.3 Explaining DT-Net Predictions

An important feature of our DT-Net approach is that human understandable explanations can be easily derived from DT-Net predictions. We first prove several important properties about threshold functions that we will use to derive explanations from them. We then describe how explanations can be derived in the single threshold function case, followed by a discussion regarding how explanations can be derived from the overall DT-Net. All proofs to theoretical results in this section can be found in the supplementary material.

2.3.1 Threshold Functions and Primes

A feed-forward neural network typically comprises layers of neurons. Further, a neuron with binary inputs and full-precision weights performs the following computation:

$$f(\mathbf{x}) = \varphi\left(\mathbf{w}^T\mathbf{x} + b\right), \tag{2.2}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a weight vector $\langle w_1, w_2, \ldots, w_n \rangle$, $\mathbf{x} \in \mathbb{R}^n$ is an input vector $\langle x_1, x_2, \ldots, x_n \rangle$, $b \in \mathbb{R}$ is a bias term, and $\varphi(\cdot)$ is a non-linear activation function. Common activation functions include ReLU activation, the sigmoid function, and the step function.

When the n inputs are binary features, and the step function is used for activation, the neuron $f(\mathbf{x})$ corresponds to a *threshold function*[2], where

$$\begin{aligned} z(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b, & (2.3) \\ f(\mathbf{x}) &= \begin{cases} 1 & \text{if } z(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise.} \end{cases} & (2.4) \end{aligned}$$

A threshold function f implements an underlying Boolean logic function $f : \{0,1\}^n \to \{0,1\}$. As such, terminologies and properties from Boolean algebra apply.

An *instance* $\alpha \in \{0,1\}^n$ is a specific assignment to the input features. With respect to the threshold function f, a positive instance is one such that $f(\alpha) = 1$, and a negative instance is one such that $f(\alpha) = 0$. A *literal* ℓ_i is a feature (positive literal) or its negation (negative literal), denoted as $\ell_i = x_i$ and $\ell_i = \bar{x}_i$, respectively. A *term* π is a consistent conjunction of literals, e.g., $x_1 \wedge \bar{x}_2 \wedge x_3$, or simply $x_1\bar{x}_2x_3$[3]. The *length* of π, denoted as $|\pi|$, is the number of literals that it includes. We say that a term π_i *covers* or *contains* another term π_j, written as $\pi_j \Rightarrow \pi_i$, if and only if π_j includes all the literals in π_i (e.g., $x_1\bar{x}_2$ covers $x_1\bar{x}_2x_3$).

An *implicant* π of a Boolean function f is a term that *satisfies* f, written as $\pi \Rightarrow f$, meaning all instances covered by π are positive instances. A *prime implicant* (or simply a *prime*) is an implicant that is not covered by any other implicant. A prime is *essential* if it covers an instance that is not covered by any other prime. A set of primes $\{\pi_1, \ldots, \pi_m\}$ is a *prime cover* for f if $\vee_{i=1}^m \pi_i$ is equivalent to f, and it is a *prime and irredundant cover* if no prime π_i can be removed from $\{\pi_1, \ldots, \pi_m\}$ such that the set remains a prime cover.

Several concepts are introduced next to prove several important properties about deriving prime implicants from threshold functions.

Definition 1 (Slack)**.** The *slack* of an instance α with respect to a threshold function f corre-

[2] A threshold function is also commonly written in the form $\mathbf{w}^T\mathbf{x} \geq \theta$, which is equivalent to $\mathbf{w}^T\mathbf{x} - \theta \geq 0$, where θ is referred to as the *threshold*. This is equivalent to Equations 2.3 and 2.4, with $\theta = -b$. We will use the form expressed in Equations 2.3 and 2.4, as this is the common expression form for describing neurons.

[3] As a shorthand, $[101]$ is used to denote the term $x_1\bar{x}_2x_3$. Here is another shorthand example: $[10-]$ is used to denote the term $x_1\bar{x}_2$, where "$-$" means the corresponding feature is excluded from the term.

sponds to $z(\alpha)$ in Equation 2.3. Therefore,

$$f(\alpha) = \begin{cases} 1 & \text{if the slack is non-negative} \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

The *slack* of a term π is defined as the minimum slack among the instances that π covers:

$$z(\pi) = \min_{\alpha} z(\alpha), \quad \text{s.t. } \alpha \Rightarrow \pi. \tag{2.6}$$

Note $z(\pi)$ can be directly computed by setting every feature x_i that does not appear in the term π to its worst-case value, which minimizes $z(\pi)$: i.e., if $w_i > 0$, set $x_i = 0$; otherwise, set $x_i = 1$.

Definition 2 (Maximum Slack)**.** We define the *maximum slack* of a threshold function f to be the largest slack among all possible assignments. In other words,

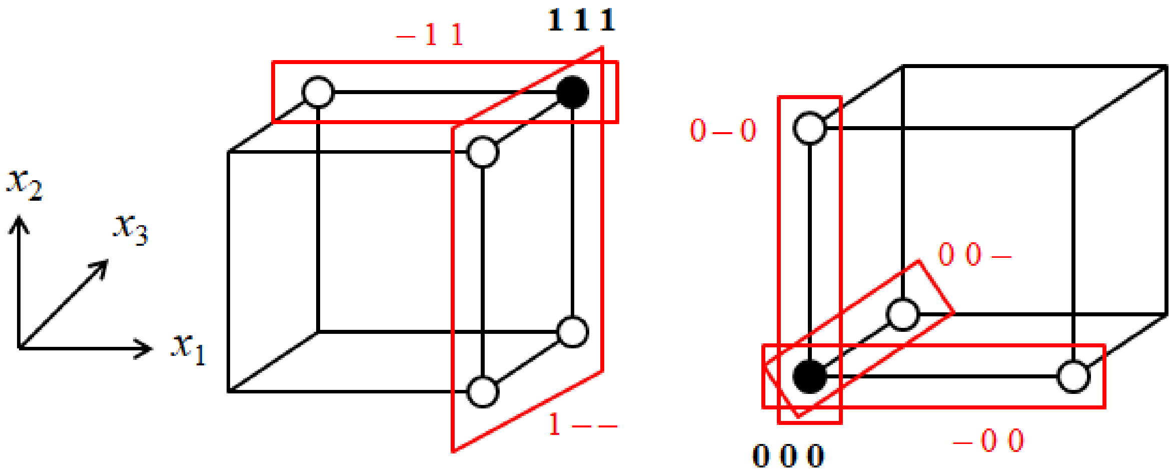
$$z_{max} = \max_{\alpha} z(\alpha), \quad \forall \alpha \in \{0,1\}^n. \tag{2.7}$$

This maximum slack can be directly computed by setting a feature x_i to its best-case value to maximize $z(\alpha)$ if it appears in f with a non-zero weight: i.e., set $x_i = 1$ if $w_i > 0$ and $x_i = 0$ otherwise.

Definition 3 (Base Term)**.** For a threshold function f, we define the *base term* π_{base} to be a term that includes the literal x_i if $w_i > 0$ and the literal $\bar{x}_i$ if $w_i < 0$ (no literal for x_i is included if $w_i = 0$).

Proposition 1. *The base term always achieves the maximum slack. In other words,* $z(\pi_{base}) = z_{max}$.

Next, we illustrate the above definitions with two examples, as depicted in Figure 2.2. Consider the first example depicted in Figure 2.2b The base term is $[111]$ as it achieves the



(a) $2.1x_1 + x_2 + x_3 - 2 \geq 0$. **(b)** $-x_1 - x_2 - x_3 + 1 \geq 0$.

Figure 2.2. Threshold function examples with the corresponding base terms (e.g., 111) and primes (e.g., $1--$).

maximum slack of $2.1+1+1-2 = 2.1$. The base term is shown in a black circle, while the remaining positive instances are shown in white circles. There are two primes in this example (shown in red). One prime $[-11]$ can be derived by *expanding* the base term $[111]$ in the x_1 direction (by removing the literal x_1), whereas the other prime $[1--]$ can be derived by expanding the base term $[111]$ in both the x_2 and x_3 directions. In both primes, the slack of each is non-negative, but removing one more literal would cause the corresponding slack to become negative.

Intuitively, all primes can be generated *from* the base term π_{base} with the maximum slack. If there exists a non-zero w_i such that $|w_i| \leq z_{max}$, then the corresponding literal for x_i can be removed from π_{max} to produce an intermediate implicant $\tilde{\pi}$. This process can be repeated by removing each additional literal as long as there is a corresponding non-zero w_i such that $|w_i| \leq z(\tilde{\pi})$, the remaining slack, until a prime is produced.

In the second example depicted in Figure 2.2b, the base term is $[000]$ as it achieves the maximum slack of $0-0-0+1 = 1$. There are three primes in this example, corresponding to expanding in each of the three directions, to produce $[-00]$, $[0-0]$, and $[00-]$.

Theorem 2. *All primes of a threshold function cover the base term.*

Proof. We prove this by contradiction. Assume a prime π does not cover the base term. Then, there must exist a literal $\ell_i \in \{x_i, \bar{x}_i\}$ that is in π but not in π_{base}. Consider the following two cases. First, if $\bar{\ell}_i$ is present in π_{base}, then ℓ_i corresponds to the worst-case value and removing it from π will increase the slack. Second, if π_{base} does not include ℓ_i or $\bar{\ell}_i$, then it implies $w_i = 0$ and removing ℓ_i from π does not change the slack. In both cases, since π is a prime, we have $z(\pi) \geq 0$, and hence $z(\pi \setminus \{\ell_i\}) \geq 0$, which is contradictory to the definition of primes. □

Theorem 3. *All primes of a threshold function are essential.*

Proof. We prove this by contradiction. Assume a prime π_1 is not essential, which means there exists an instance covered by π_1 that is also covered by another prime. Consider an instance α covered by π_1 that disagrees with every literal ℓ_i that is in π_{base} but not in π_1. Suppose α is covered by another prime π_2, implying that for every such ℓ_i, π_2 either includes $\bar{\ell}_i$ or excludes both ℓ_i and $\bar{\ell}_i$. According to Theorem 2, we have $\pi_{base} \Rightarrow \pi_1$ and $\pi_{base} \Rightarrow \pi_2$. Since ℓ_i is covered by π_{base}, we must have π_2 excludes every such ℓ_i and $\bar{\ell}_i$, which implies all literals removed from π_{base} to produce π_1 are also removed to produce π_2. As a result, $\pi_1 \Rightarrow \pi_2$. This means either $\pi_1 = \pi_2$ or π_1 is not a prime, which are contradictory in both cases. □

Corollary 4. *The prime cover of a threshold function is unique and irredundant.*

Proof. It follows from the proof of Theorem 3. □

2.3.2 Explaining a Single Threshold Function

We first consider the problem of deriving an explanation for the single threshold function case. A threshold function f is equivalent to a Boolean *classifier*, where $f(\alpha) = 1$ means the *decision* is positive, and $f(\alpha) = 0$ means the decision is negative. For a positive prediction, an explanation can be thought of as some subset of its literals. Referring to the example depicted in Figure 2.1, an explanation why an individual is at high heart disease risk may be that the individual is *older than 50* and has *high cholesterol*. Another explanation may be that the individual is a *smoker*. We formalize below what explanations are and how they can be readily derived in the case of a single threshold function.

Definition 4 (Explanation)**.** An *explanation* for a positive decision on an instance α is an implicant that contains the instance.

Definition 5 (Minimal Explanation)**.** A *minimal explanation* is a prime that contains the instance.

Definition 6 (Simplest Explanation)**.** A *simplest explanation* a shortest length minimal explanation.

Note that minimal and simplest explanations are not unique. As shown in [82], for a threshold function f and a positive instance α, finding minimal explanations corresponds to finding prime implicants[4] of f that contain α. The prime associated with a minimal explanation corresponds to a minimal subset of features that are sufficient for the positive prediction. This can be achieved by first converting the threshold function f into a logic representation, followed by using known prime generation algorithms to generate all minimal explanations, where the simplest explanation (shortest prime containing α) can be found, but this approach is worst-case exponential in time and space. Fortunately, the simplest explanation can be directly derived from the threshold function f, as discussed below.

Definition 7 (Base Explanation)**.** Given a threshold function f as a classifier and a positive instance α, we define the *base explanation*, written as $\pi_{\textit{base-exp}}$, to be the *supercube* of the base term π_{base} and the instance α, written as $super(\pi_{base}, \alpha)$.

The *supercube* of two terms, $super(\pi_i, \pi_j)$, is a new term derived by removing literals from π_i that do not appear in π_j. The operation is symmetric in that the new term can also be derived by removing literals from π_j that do not appear in π_i.

[4] In [82], minimal explanations are referred to as PI-explanations, where PI refers to prime implicants. The work in [82] was developed for Naive Bayes Classifiers, but the same PI-explanation concept also applies to threshold functions. In [36], PI-explanations are referred to as abductive explanations.

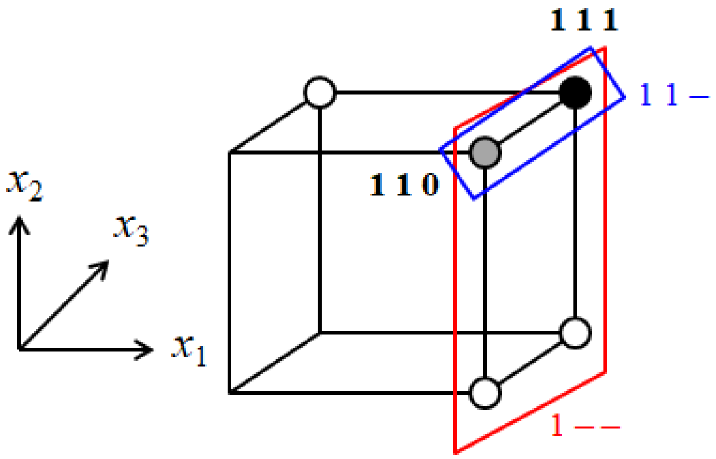


Figure 2.3. An example of base and minimal explanations, with f defined as $2.1x_1 + x_2 + x_3 - 2 \geq 0$ and $\alpha = [110]$.

Theorem 5. *The set of minimal explanations of a positive instance for a threshold function includes only essential primes.*

Proof. It follows from the proof of Theorem 3. □

Theorem 6. *All minimal explanations of a positive instance for a threshold function cover the base explanation.*

Proof. We prove this by contradiction. Assume a minimal explanation π has a literal ℓ_i that the base explanation $\pi_{base\text{-}exp}$ does not have. Then there are two possibilities. The first is that $\pi_{base\text{-}exp}$ has the literal $\bar{\ell}_i$ instead of ℓ_i. Based on the definition of the base explanation, the instance α must also have $\bar{\ell}_i$. Since ℓ_i and $\bar{\ell}_i$ cannot both appear in π, π does not have $\bar{\ell}_i$ and thus does not contain instance α. This contradicts the definition of an explanation. The second possibility is that $\pi_{base\text{-}exp}$ does not have ℓ_i or $\bar{\ell}_i$. Since π is an explanation of α, π contains α and thus α also has the literal ℓ_i. Then, we know that the base term of the threshold function must have $\bar{\ell}_i$, so that $\pi_{base\text{-}exp}$, as a supercube of π_{base} and α, does not have ℓ_i or $\bar{\ell}_i$. According to the definition of the base term, the corresponding weight w_i of the threshold function is negative. At this point, it is obvious that a new term $\pi \setminus \{\ell_i\}$ is still a valid explanation because removing ℓ_i from π does not change the slack of π, which is contradictory to the premise that π is a minimal explanation. □

Consider the example depicted in Figure 2.3. In this example, $\pi_{base} = [111]$ and $\alpha = [110]$ (shown as a gray circle), then the base explanation is $[11-]$ (shown in blue). The generation of explanations can be performed in a similar way as prime generation. According to Theorem 6, all minimal explanations, which are primes containing the instance α, can be generated *from* the base explanation $\pi_{base\text{-}exp}$ with the available slack $z(\pi_{base\text{-}exp})$. Consider again the example depicted in Figure 2.3. The base explanation $[11-]$ can be expanded into a minimal explanation by *expanding* in the x_2 direction (by removing the literal x_2) to obtain the prime and minimal explanation $[1--]$. There are no other minimal explanations, making $[1--]$ also the simplest explanation. In particular, if there exists a non-zero w_i such that $|w_i| \leq z(\pi_{base\text{-}exp})$, then the corresponding literal for x_i can be removed from $\pi_{base\text{-}exp}$ to produce an intermediate implicant. This process can be repeated as long as there is a corresponding non-zero w_i such that $|w_i|$ is less than or equal to the remaining slack, until a minimal explanation is produced.

Based on this intuition, we propose the smallest-absolute-weights-first removal algorithm, which is summarized in Algorithm 1. This is a very fast and simple greedy algorithm that can guarantee the simplest explanation.

Algorithm 1. Smallest-absolute-weights-first removal

```
Input: Threshold function $f$, base explanation $\pi_{base\text{-}exp}$
Output: Simplest explanation $\pi$

$L \leftarrow \{\ell_i \in \pi_{base\text{-}exp}\}$ sorted by $|w_i|$ in ascending order
$\pi \leftarrow \pi_{base\text{-}exp}$
for $\ell_i \in L$ do
    if $z(\pi \setminus \{\ell_i\}) \geq 0$ then
        $\pi \leftarrow \pi \setminus \{\ell_i\}$
    else
        break
    end if
end for
return $\pi$
```

Theorem 7. *Algorithm 1 finds a simplest explanation for a positive instance of a threshold function.*

Proof. We prove this by contradiction. Assume π_1 is the explanation generated by Algorithm 1 and π_2 is a shorter explanation. According to Algorithm 1, π_1 is a minimal explanation (prime) since further removing any literal from π_1 would cause its slack to become negative. Consider two sets of literals $\{\ell_i \mid \ell_i \in \pi_2, \ell_i \notin \pi_1\}$ and $\{\ell_j \mid \ell_j \in \pi_1, \ell_j \notin \pi_2\}$. Since π_2 is shorter than π_1, we have $|\{\ell_i \mid \ell_i \in \pi_2, \ell_i \notin \pi_1\}| < |\{\ell_j \mid \ell_j \in \pi_1, \ell_j \notin \pi_2\}|$. For π_2, keep replacing such ℓ_i with ℓ_j until there does not exist such ℓ_i and denote by π_3 the produced term. Since π_1 is generated by Algorithm 1, we always have $w_i > w_j$ for w_i and w_j corresponding to any combinations of ℓ_i and ℓ_j, respectively. Therefore, we must have $z(\pi_3) \geq 0$ and π_3 is an implicant. Further, we have $\pi_1 \Rightarrow \pi_3$, which is contradictory to the premise that π_1 is a prime. □

2.3.3 Explaining the Disjunctive Threshold Network

We next consider the problem of deriving an explanation for the entire disjunctive threshold network. Since all threshold functions are combined using a logical OR operator, an explanation of a positive instance for one of the activated threshold functions is also an explanation for the whole network. Therefore, we can simply enumerate Algorithm 1 on each of the activated threshold functions and return the shortest explanation among them as an explanation for the overall network. This enumeration algorithm is also very fast and simple, as depicted in Algorithm 2.

2.4 Simplifying Explanations Through Sparsity-Inducing Regularization

DT-Net can be accurately trained using well-developed stochastic gradient descent training algorithms. We use a binary cross-entropy loss function at the output, and we use a straight-through estimator with gradient clipping [5] in the hidden layer to backpropagate gradient updates through the binary step activations.

It should be clear from the previous section that zero weights in a threshold function mean that the corresponding inputs will not have any effect on the logic of the threshold

Algorithm 2. Deriving explanations from DT-Net

Input: Set of threshold functions $F = \{f_1, f_2, \dots, f_n\}$, positive instance α
Output: DT-Net explanation $\tilde{\pi}$

```
$S \leftarrow \{\}$
**for** $f_i \in F$ **do**
    **if** $f_i(\alpha) = 1$ **then**
        $\pi_{\textit{base-exp}} \leftarrow$ get base explanation of $f_i$
        $\pi \leftarrow$ Algorithm 1$(f_i, \pi_{\textit{base-exp}})$
        $S \leftarrow S \cup \{\pi\}$
    **end if**
**end for**
$\tilde{\pi} \leftarrow \arg\min_{\pi \in S} |\pi|$
**return** $\tilde{\pi}$
```

function, which means those input features can be removed from any explanation derived from that threshold function. Therefore, promoting the sparsity of hidden layer threshold functions indirectly simplifies explanations. Further, as shown in [31], neurons with zero input connections (meaning all its weights are zero) can be safely removed since these dead neurons will have no effect on the output classification. Besides a training strategy that maximizes the number of zero weights, encouraging weights with small absolute magnitudes is also beneficial in deriving simpler explanations. This is because more input features can be removed from an explanation if the corresponding weights have small absolute values relative to the available slack.

We can encourage sparsity by including a regularization term into the overall loss function of the form

$$\mathscr{L} = \mathscr{L}_{BCE} + \lambda \mathscr{L}_R(W), \tag{2.8}$$

where $\mathscr{L}_{BCE}$ is the binary cross-entropy loss, $\mathscr{L}_R(\cdot)$ is the regularization loss over the weight matrices W in the network, and λ is the regularization coefficient. Fortunately, we can encourage both zero weights and weights with small values in absolute magnitude by means of sparsity-inducing regularization. In particular, we use the reweighted L_1 regularization [13] approach that penalizes smaller absolute value weights so that they are driven towards zero faster, resulting in more weights near zero. We also incorporate a pruning method [31] to eliminate weights with absolute magnitudes below a certain threshold. Weights near this threshold that remain tend to be small so that they are more likely to be eliminated in our algorithms to derive explanations. As shown in [13], a log-sum penalty term,

$$\mathscr{L}_R(W) = \log(\|W\|_1 + \varepsilon), \tag{2.9}$$

can be used to achieve reweighted L_1 minimization, where $\varepsilon > 0$ is a small value (e.g., $\varepsilon = 0.1$) added to ensure numerical stability. As shown in the evaluation section, this sparsity-inducing regularization approach not only simplifies the explanations, but it also leads to the removal of many dead neurons.

2.5 Evaluation

Table 2.1. Average test accuracy and complexity (in parentheses) based on the 5-fold cross-validation.

dataset	DT-Net	DR-Net	CG	RIPPER	BRS	CART	RF	XGB
adult	**83.64** (11.83)	82.55 (11.46)	82.60 (6.54)	82.25 (5.16)	78.78 (3.00)	82.44 (13.11)	84.03	84.41
magic	**85.34** (4.70)	83.91 (2.73)	83.33 (2.82)	82.86 (4.55)	81.37 (3.00)	83.18 (11.98)	86.71	87.16
house	**87.61** (10.43)	86.07 (3.56)	83.80 (2.90)	81.43 (5.74)	83.26 (3.00)	85.10 (12.22)	88.49	88.92
recidivism	**65.39** (6.31)	64.09 (2.06)	64.57 (2.98)	64.84 (4.22)	61.98 (3.00)	62.85 (9.86)	66.77	64.33
chess	**89.30** (7.28)	84.47 (7.70)	81.93 (6.97)	85.46 (9.80)	74.66 (3.00)	85.36 (16.02)	92.63	94.98
retention	**93.47** (3.64)	87.78 (3.23)	90.77 (3.68)	88.92 (3.73)	89.37 (3.00)	90.11 (11.86)	93.43	94.29
churn	**79.51** (8.92)	78.85 (6.80)	79.21 (2.71)	78.27 (4.81)	76.74 (3.00)	79.00 (9.82)	80.35	77.45
airline	**94.41** (4.71)	93.32 (3.45)	90.10 (3.50)	93.08 (4.28)	90.71 (2.90)	90.21 (12.64)	94.79	95.94

Benchmarks. The numerical experiments were evaluated on 8 publicly available binarized classification datasets, most of which have more than 10,000 instances and comprise categorical and numerical attributes for each instance before binarization. We used three datasets from the UCI Machine Learning Repository [27], namely adult (Adult Census), magic (MAGIC Gamma Telescope), and chess (Chess: King-Rook vs. King). Two of the selected datasets are from Kaggle: churn (Telco Customer Churn) and airline (Airline Passenger Satisfaction). The other three datasets are: house (House_16H) [89], retention (TED Dataset) [3], and recidivism (Predicting Recidivism) [79]. These datasets were shuffled (with a fixed seed to ensure the consistency for all approaches) and split into 5 sets of training and test datasets using 5-fold cross-validation. All experimental results are derived by running the classifiers on 5 test sets and averaging the results.

DT-Net Configurations. For DT-Net, we used the Adam optimizer with a fixed learning rate of 10^{-2} and no weight decay across all experiments. There are 100 neurons in the hidden layer to ensure there is an efficient search space for all datasets, and the network is trained for 1,000 epochs to guarantee complete convergence. For simplicity, the batch size is fixed at 2,000 and the weights are uniformly initialized within the range between 0 and 1. Other parameters were selected according to the nested 5-fold cross-validation, which will be discussed in the following subsections.

2.5.1 Classification Performance

Baselines and Pre-processing. In this evaluation, we compare our approach with four rule learners, including Decision Rule Net (DR-Net) [53], the Column-Generation-Based algorithm (CG) [25], RIPPER [20], and Bayesian Rule Sets (BRS) [91]. We also compare our approach with decision trees (CART), random forests (RF), and gradient boosting trees (XGB). RIPPER is a greedy rule mining approach based on sequential covering. DR-Net, BRS and CG are recent rule-set-generation classifiers that optimize both for accuracy and interpretability, and CART [9] is a decision tree learning algorithm. We use random forest (RF) [10] and XGBoost (XGB)

[15] to provide baselines for typical performances that black-box models can achieve on the datasets evaluated. For all datasets, we adopted the pre-processing approach discussed in [53] to binarize numerical and categorical features. BRS and CG do not directly consider the negation of features. Therefore, we followed the procedures described in their papers to append the negations of binarized features so that they can be considered in their rule sets.

Complexity Measurement. For our model and other interpretable models, the classification performance was evaluated using both accuracy and interpretability. The accuracy was evaluated on the test set and the interpretability was measured by the average explanation complexity. We note that while rule learners generally consider the complexity of generated rules, our model carries out the prediction without pre-learning any rules, but derives the explanation afterwards. Therefore, we proposed a new complexity metric, namely explanation complexity, as the average length of explanations for all positive instances in the test dataset. For DT-Net, the explanations were produced according to the algorithm discussed in Section 2.3.3 and therefore the complexity is the length of the explanation. For rule learners, the complexity was computed based on the simplest rule that covers the test instance. For CART, the explanation was derived by tracing down the decision path from the root node, and the complexity is measured by the number of nodes in the decision path.

Parameter Tuning. We evaluated the predictive performance of DT-Net by comparing both test accuracy and complexity with other state-of-the-art machine learning models. For parameter selection in all models, we used a 5-fold nested cross-validation to improve training accuracy. Specifically, the best accuracy is achieved by tuning the regularization coefficient λ for DT-Net, minimum number of samples per leaf for CART and RF, and the regularization term for XGBoost. We tuned the same parameters mentioned in [53] for DR-Net, CG, RIPPER and BRS. We take the average performance over the 5 training-testing pairings as the final reported results. We summarize the accuracies of all models and the complexities of the interpretable models in Table 2.1, where the best accuracies among interpretable models are highlighted in bold.

As can be observed in Table 2.1, DT-Net achieves the best accuracy among all interpretable models on all datasets, indicating that DT-Net has a significant advantage over other interpretable models on generalization capability. At the same time, DT-Net achieves an accuracy very close to the uninterpretable models on most datasets (except for the chess dataset) and it even outperforms them in some cases (see churn and airline datasets). Moreover, since DT-Net is a neural network, the performance of DT-Net can possibly be further improved with finer parameter tuning and more advanced training techniques. Regarding complexity, it can be seen that though DT-Net does not produce the simplest explanations, it still shows admissible interpretability in that its complexity is within the same magnitude of other approaches. In particular, we note that DT-Net always outperforms the traditional decision tree on all datasets and thus in real-world applications, DT-Net can generally substitute decision trees with both higher accuracy and lower complexity.

2.5.2 Effects of Sparsity-Inducing Regularization

In our experiments, the networks are composed of a large number of threshold functions, e.g. 100 neurons in the hidden layer, to ensure enough capacity. Simplifying the neural network using the regularization and pruning methods mentioned earlier helps reduce both the complexity and the computation time of the explanations. We show in Table 2.2 that our neural network achieves very high sparsity, which partially explains why our explanation generation procedure has relatively low computational cost.

As can be observed from Table 2.2, most threshold functions are disabled after pruning, which verifies the effectiveness of our regularization method in excluding the redundant capacity. Further, the remaining neurons generally achieve an average sparsity over 50%. This means that more than half of the literals are directly removed before applying our algorithms, which explains how we remove most literals and generate explanations with reasonable complexity in our experiments. In addition, individual positive instances in general are only produced by about 2 or 3 neurons for all datasets. The fact that each instance only activates a few neurons explains

Table 2.2. Statistics of DT-Net after pruning. # pruned: the number of neurons pruned in the hidden layer. Sparsity: the percentage of the zero weights in each neuron. # activated: the number of activated neurons for each test instance. The average over 5 partitions is reported for every dataset (standard deviation in parentheses).

dataset	# pruned	Sparsity	# activated
adult	96.2	81.21%	1.49 (0.68)
magic	87.8	82.81%	2.84 (1.63)
house	90.2	70.95%	3.63 (1.78)
recidivism	95.0	76.00%	1.12 (0.36)
chess	62.2	58.50%	2.75 (1.82)
retention	67.8	80.27%	3.91 (2.77)
churn	96.8	70.90%	1.34 (0.51)
airline	87.2	87.92%	4.48 (2.03)

why our explanation generation algorithm generally produces explanations that are minimal for the network.

2.6 Concluding Remarks

We proposed in this work a neural network architecture called DT-Net for tabular data classification that provides both high accuracy performance and interpretability. An important feature of the proposed solution is that only a simple greedy algorithm is required to provide an explanation with the prediction that is human-understandable. We further employ a sparsity-inducing regularization approach to sparsify the threshold functions so that the derived explanations are simple. In comparison with other explainable decision models, our evaluation shows that our proposed approach can achieve superior predictive performance on a broad set of tabular data classification datasets.

2.7 Acknowledgements

Chapter 2, in full, is a reprint of the material as it appears in IEEE Open Journal of the Computer Society, 2023. Weijia Wang, Litao Qiao, and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Tabular machine learning using conjunctive threshold neural networks

3.1 Introduction

In machine learning applications like healthcare and criminal justice where human lives may be deeply impacted, creating decision models that can provide human understandable explanations is critically important [77]. In these applications, the datasets are often provided as tabular data with samples as rows in a table and a common set of naturally meaningful features as columns. A toy example of a tabular dataset is shown in Table 3.1.

Table 3.1. A toy example of a tabular dataset. The first seven columns are input features, and the last column is the classification.

Gender	Age	BP	Cholesterol	Glucose	Smoker	Drinker	Disease
Male	34	Normal	Normal	Normal	No	Yes	No
Female	62	High	Normal	High	Yes	No	Yes
Male	55	High	High	Normal	No	No	Yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Female	25	High	Normal	Normal	Yes	No	No

Due to their inherent explainability, decision rule sets [20, 47, 91, 25] are often a popular model class in these tabular learning problems. Decision rule sets not only provide accurate predictions, but the corresponding matching rules also provide explanations that humans can easily understand. In particular, the explanations are expressed directly in terms of meaningful categorical (e.g., Cholesterol equal to Normal or High) or numerical (e.g., age) input attributes, where the binary encoding of categorical and numerical attributes is well-studied[91, 25]. However, they are not the winning model class in these domains in terms of prediction accuracy. For example, gradient boosted and ensemble decision trees [15, 44, 10] and neural network models [43, 2] are generally superior in prediction performance. However, these models are generally considered to be black-box models [77] where the predictions are difficult or impossible to interpret. This is in sharp contrast to the interpretability of rule-based sentences that decision rule sets provide, which can be easily understood by humans.

In this paper, we propose a new neural network architecture for tabular data called a Conjunctive Threshold Neural Network, or CT-Net for short. The proposed structure comprises a hidden layer of threshold logic functions, which are just conventional neurons with a step activation function that are trainable with arbitrary (positive or negative) full-precision weights and biases. This neural network architecture can be trained to achieve high prediction accuracy, but unlike conventional neural network models, gradient boosting decision trees, and random decision forests, human understandable explanations in terms of meaningful input features similar to decision rules can be easily derived from CT-Net. Also, unlike existing interpretable rule-learning methods [91, 25, 71] that provide human understandable explanations, we do not ever explicitly generate a decision rule set from CT-Net. This means that our network of threshold functions can implicitly encode potentially complicated rules to achieve high prediction accuracy, but yet the explanations generated can nonetheless be understandable. In particular, the explanation derived is provably *minimal* in the number of features in the conjunction. Therefore, we believe CT-Net can be widely used as an replacement for tree ensemble methods (random forest and gradient boosting trees) in the area where both accuracy and interpretability are required.

The outline of the paper is as follows: Section 3.2 summarizes related work. Section 3.3 describes our proposed CT-Net architecture. Section 3.4 describes how provably minimal explanations can be easily derived from a CT-Net inference. Section 3.5 describes a sparsity-promoting regularization approach for training CT-Nets. Section 3.6 provides extensive evaluation of our proposed approach. Section 3.7 concludes the paper.

3.2 Related Work

Besides decision rule sets [20, 47, 91, 25], decision lists [74, 48] and decision trees [9] are also interpretable rule-based models. Decision lists are ordered rules in an if-then-else sequence, and decision trees have paths that can be interpreted as rules. In addition to providing prediction, these methods also provide human understandable explanations that can be derived from the matching rule.

Gradient boosting decision trees [15, 44] and random forests [10] have also been used to provide better predictions in tabular data classification problems. Although these methods provide superior prediction performance in comparison to rule-based methods, they are generally not interpretable. In certain application areas, their lack of interpretability may make it difficult to gain public trust for their use, which may hinder their widespread adoption in these domains.

Building on the notable success that deep neural networks have shown on perceptual learning tasks, like image classification [32], researchers have recently turned to neural network models for tabular data learning as well [71, 43, 2]. The work in [43, 2] aim to capture aspects of gradient boosting decision trees and random forests that have made these models successful, and they are able to achieve comparable performance as these approaches with neural models. However, like gradient boosting decision trees and random decision forests,these models remain uninterpretable in the sense that they do not provide explanations that are easily understandable by humans.

In contrast, [71] recently proposed a neural network architecture that directly maps to a decision rule set in disjunctive normal form. In this architecture, the neurons in the hidden layer are restricted in a way so that they map directly to a conjunction (AND) of input features that correspond to interpretable decision rules, followed by an output neuron that maps to a disjunction

(OR) operation that aggregates a collection of decision rules into a set. This approach achieves better performance than traditional rule-based and decision tree methods [20, 47, 91, 25, 9] while retaining the ability to provide human understandable explanations. However, the restrictions imposed on the hidden layer neurons to have direct one-to-one mappings to conjunctive rules unnecessarily limits the search space during neural net training.

Another body of work aims to develop post-hoc explainers that can explain predictions from black-box models. Heuristic algorithms are employed in [73, 72] to generate explanations without the knowledge of the entire model. Although a primary objective of these algorithms is to achieve high fidelity of the explanations to the original model, the derived explanations cannot be guaranteed to be completely consistent with the underlying model distribution.

Finally, the works in [82, 81, 18, 36, 40, 4] are on the compilation of models into tractable forms. In these approaches, explanations consistent with the original model can be queried from the model's equivalent tractable form. Our work is complementary in that we aim for an approach in which a simple and fast algorithm can be applied to directly derive human understandable explanations from our proposed model.

3.3 Conjunctive Threshold Neural Networks

In this section, we introduce the architecture of the proposed Conjunctive Threshold Neural Network, or CT-Net for short. The network is designed for tabular classification problems where besides making accurate predictions, the *explanation* of decisions is also essential. In particular, CT-Net is a simple three layer neural network architecture, comprising an input layer of n input units, a hidden layer of k units, and an output layer with a single output unit. A toy example is shown in Figure 3.1 for predicting heart disease risk, which we use to illustrate several key ideas in this section.

Input layer: The input layer consists of n input units, each passing its corresponding assigned binarized value to each neuron in the hidden layer. Tabular datasets often comprises binarized,

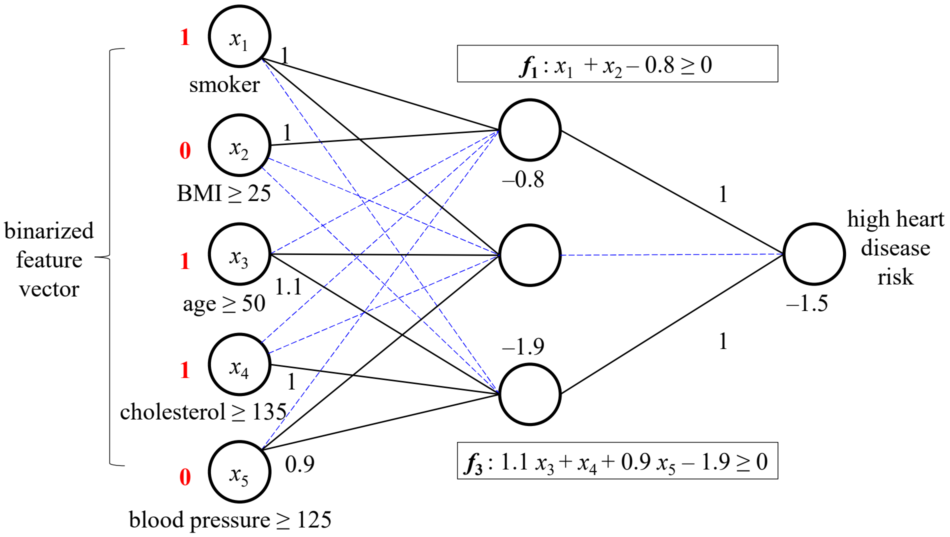


Figure 3.1. A toy example of the CT-Net architecture with hidden layer neurons as threshold logic functions and the output neuron implementing a conjunction.

categorical and numerical attributes. To handle categorical and numerical attributes, well-studied pre-processing procedures in the machine learning literature can be used to encode them into binarized input vectors[1]: standard one-hot encoding can be used for categorical attributes, and standard quantile discretization can be used for numerical attributes.

Hidden layer of threshold functions: The hidden layer comprises k neurons that are trainable with arbitrary (positive or negative) full-precision weights and biases. They implement threshold functions by means of a binary step activation function. In Figure 3.1, the blue dashed lines at the inputs of a hidden neuron indicate that the corresponding features have zero weights, which means they do not appear in the corresponding threshold function. As discussed in the next section, each threshold function implicitly implements an underlying Boolean logic function that encodes logical conditions on the inputs that will lead to a positive prediction.

[1] Interpretable rule-learning methods [91, 25, 71] widely studied to model tabular classification problems also commonly assume this input binarization pre-processing step.

Output conjunction layer: The output unit implements a *conjunction* of a subset of the k threshold functions in the hidden layer, which is also implemented as a single threshold function with trainable binarized weights (more details are discussed in Section 3.5), where a weight of 1 or 0 indicates if the corresponding threshold function in the hidden layer is included in or excluded from the conjunction, respectively (a 0 weight is shown as a blue dashed line at the input of the output neuron in Figure 3.1). Further, a dynamic bias based on the weights is employed as follows:

$$b = -\sum_{i=1}^{k} w_i + \varepsilon. \tag{3.1}$$

where ε is a small number between 0 and 1 (e.g., $\varepsilon = 0.5$), and w_i is the binary weight (i.e., 1 or 0) as just discussed. This output threshold unit implements a logical-AND operation since the output unit can make a positive prediction (with the logits of ε) only if all threshold functions (that are not excluded with a zero weight) in the hidden layer are activated, whereas by default, it makes a negative prediction due to the negative summation of the weights. Since each threshold function implicitly implements an underlying Boolean logic function, the logical-AND of these threshold functions also implicitly implements a Boolean logic function for the network. This conjunction of threshold functions can be trained to implement any Boolean logic function at the output, which can model any prediction problem that can be encoded into a binary classifier.

Straight-through estimator: As discussed earlier, the proposed neural network incorporates the binary step activation function, which is almost non-differentiable everywhere. Therefore, we adopt the straight-through estimator [5] to address this issue. In particular, it was originally proposed in [5] to use the identity function as the derivative of a step function, while [21] further incorporates the gradient clipping technique to cancel the gradients when the activation is too large, i.e., the backward function is similar to ReLU, which also introduces non-linearity into the network. In our approach, we empirically found it is better to clip the gradients when the full-precision activation (pre-step activation) is either too large or too small, which is analogous to the backward function of the clipped ReLU activation. This approach in practice effectively prevents the weights from getting infinitely large or small. Mathematically, the straight-through estimator with the gradient clipping technique that we use to address this problem is as follows:

$$g_{\hat{z}_i} = \begin{cases} 0, & \text{if } z_i < -1 \text{ or } z_i > 0 \\ g_{z_i}, & \text{otherwise} \end{cases} \tag{3.2}$$

where z and $\hat{z}_i$ are the full-precision activation and the binarized activation after our step function, respectively. L is the classification loss. $g_{\hat{z}_i} = \frac{\partial L}{\partial \hat{z}_i}$ and $g_{z_i} = \frac{\partial L}{\partial z_i}$, which are the gradients of classification loss w.r.t. $\hat{z}_i$ and z_i, respectively.

Similar to the clipped ReLU activation function with a clipping boundary of 1, the outputs produced by the step function always fall into the range between 0 and 1. Therefore, our step activation function can be essentially viewed as a low-precision clipped ReLU function. However, we note that a floating-point 0 will be directly binarized to 1 in our approach, which corresponds to a shift introduced by the binarization, which should be addressed in backward propagation. As a result, we propose to clip the gradient at -1 and 0 rather than 0 and 1 as in the clipped ReLU activation, and experimental results show this straight-through estimator work very well in practice.

An example: Consider again the toy example shown in Figure 3.1 for predicting heart disease risk. The input variables $x_1, x_2, \ldots, x_5$ correspond to whether or not the individual is a smoker, overweight, or older than 50, or has high cholesterol or blood pressure, respectively. The instance shown (in red) is the input assignment $\langle x_1, x_2, x_3, x_4, x_5 \rangle = [10110]$. With this instance, threshold functions f_1 and f_3 evaluate to true (i.e., evaluate to 1), whereas threshold function f_2 is directly turned off by the output layer. For f_1, the threshold function evaluates to true if either the individual is a smoker or overweight as the weights of x_1 and x_2 are both individually greater than 0.8. For f_3, the threshold function evaluates to true if any two out of the three conditions (older than 50, high cholesterol, or high blood pressure) are true, since the sum of the weights of any two conditions will be sufficient to exceed 1.9. Indeed, the classifier will make a positive prediction that this individual is at a high risk of heart diseases since both f_1 and f_3 are activated.

To guarantee CT-Net makes a positive prediction, both f_1 and f_3 need to evaluate to true at the same time (because the output neuron implements a conjunction of inputs with 1 weights), where the individual being a *smoker* suffices the first threshold function, and what sufficiently activates f_3 is that the individual is *older than 50* and has *high cholesterol*. Therefore, the *explanation* for *why* this individual is predicted to have a high heart disease risk is *smoker*, *older than 50* and *high cholesterol*. While the explanation for this prediction is unique, it is possible that there exist multiple explanations for certain positive predictions in some scenarios. As detailed later in the paper, provably minimal explanations can be readily derived for any given positive prediction.

Unlike existing rule-learning methods methods [91, 25, 71], we do not ever explicitly generate a decision rule set from the conjunction threshold network. This means that our network of threshold functions can implicitly encode potentially complicated rules to achieve high prediction accuracy. State-of-the-art stochastic gradient descent training methods can be used to achieve high prediction accuracy. Also, well-developed sparsity-promoting techniques can be invoked to simplify the network in a way that leads to succinct threshold functions, as discussed later in the paper. In the next section, we describe more formally how provably minimal explanations can be readily derived for positive predictions made using CT-Net.

3.4 Deriving Explanations

In this section, we describe how provably minimal human understandable explanations can be readily derived from a CT-Net prediction.

3.4.1 Threshold Functions and Slack

It should be clear from the previous section that a neuron in the hidden layer of the CT-Net corresponds to a *threshold function* of the form

$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b, \tag{3.3}$$

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } z(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

When the n inputs are binary features, a threshold function f implements an underlying Boolean logic function $f : \{0,1\}^n \to \{0,1\}$. As discussed, a CT-Net is a conjunction (logical-AND) of k threshold functions, $F = \{f_1, f_2, \ldots, f_k\}$. As such, a CT-Net also implements an underlying Boolean logic function $F : \{0,1\}^n \to \{0,1\}$. Therefore, Boolean algebra terminologies and properties are applicable to both individual threshold functions as well as the overall CT-Net, which we summarize here.

An *instance* $\alpha \in \{0,1\}^n$ is a specific assignment to the input features. With respect to the CT-Net F, a positive instance is one such that $F(\alpha) = 1$, and a negative instance is one such that $F(\alpha) = 0$. A *literal* ℓ_i is a feature (positive literal) or its negation (negative literal), denoted as $\ell_i = x_i$ and $\ell_i = \bar{x}_i$, respectively. A *term* π is a consistent conjunction of literals, e.g., $x_1 \wedge \bar{x}_2 \wedge x_3$, or simply $x_1\bar{x}_2x_3$ [2]. The *length* of π, denoted as $|\pi|$, is the number of literals that it includes. We say that a term π_i *covers* or *contains* another term π_j, written as $\pi_j \Rightarrow \pi_i$, if and only if π_j includes all the literals in π_i (e.g., $x_1\bar{x}_2$ covers $x_1\bar{x}_2x_3$).

An *implicant* π of a Boolean function F is a term that *satisfies* F, written as $\pi \Rightarrow F$, meaning all instances covered by π are positive instances. A *prime implicant* (or simply a *prime*) is an implicant that is not covered by any other implicant[3].

[2] We will use $[101]$ as a shorthand for the term $x_1\bar{x}_2x_3$. As another example, we will use $[10-]$ as a shorthand for the term $x_1\bar{x}_2$, with "$-$" to mean that a literal for the corresponding feature is not included in the term.

[3] The terminologies term, implicant, and primes apply to any Boolean logic function, including both the individual threshold functions f_i and the overall logic function F induced by the CT-Net.

We next describe several concepts that we will use in the algorithm for generating explanations for positive predictions of the CT-Net.

Definition 8 (Slack)**.** For an instance α, the *slack* of α with respect to a threshold function f corresponds to $z(\alpha)$ in Equation 3.3. Therefore, $f(\alpha) = 1$ if the slack is non-negative, and 0 otherwise. For a term π, the *slack* of π is defined as the minimum slack achieved by the instances that π covers:

$$z(\pi) = \min_{\alpha} z(\alpha), \quad \text{s.t. } \alpha \Rightarrow \pi. \tag{3.5}$$

The slack of π can be directly computed by setting every feature x_i to its worst-case value if it does not appear in the term π: i.e., set $x_i = 0$ if $w_i > 0$ and $x_i = 1$ otherwise. As such, $z(\pi)$ is minimized.

Definition 9 (Group Slack)**.** Let F be the Boolean logic function defined by the *conjunction* of k threshold logic functions, $\{f_1, f_2, \ldots, f_k\}$. For an instance α, let $z_i(\alpha)$ be the slack of α with respect to the corresponding threshold function f_i. Then the *group slack* of α with respect to F corresponds to the minimum among the slacks of the threshold functions:

$$z_F(\alpha) = \min_{i} z_i(\alpha). \tag{3.6}$$

Therefore, $F(\alpha) = 1$ if the group slack is non-negative, which implies the slacks of *all* individual threshold functions to be non-negative, and 0 otherwise. For a term π, the *group slack* of π is defined as the minimum slack achieved by the instances that π covers:

$$z_F(\pi) = \min_{\alpha} z_F(\alpha), \quad \text{s.t. } \alpha \Rightarrow \pi. \tag{3.7}$$

Here also, the group slack of π can be directly computed by setting every feature x_i to its worst-case value if it does not appear in the term π: i.e., set $x_i = 0$ if $w_i > 0$ and $x_i = 1$ otherwise. As such, $z_F(\pi)$ is minimized. Note that for $z_F(\pi)$ to be non-negative, the slacks of *all* individual

$z_i(\pi)$ must also be non-negative.

3.4.2 Generating Explanations

As discussed, a CT-Net F is equivalent to an underlying Boolean logic function, which can be viewed as a *binary classifier*, where $F(\alpha) = 1$ means the *decision* is positive, and negative otherwise. Intuitively, an explanation for a positive instance is some subset of its literals.

Referring to the example depicted in Figure 3.1, an explanation for the overall CT-Net requires it to be an explanation for *both* f_1 and f_3, as the CT-Net output is a conjunction of both threshold functions. In this example, being a *smoker* suffices to activate f_1, and being *older than 50* with *high cholesterol* suffices to activate f_3. Therefore, an *explanation* as to *why* this individual is predicted to have a high heart disease risk is because the individual is a *smoker* and *older than 50*, and has *high cholesterol*. We formalize below what explanations are and how they can be readily derived from a CT-Net prediction.

Definition 10 (Explanation)**.** An *explanation* for a positive decision on an instance α is an implicant that contains the instance.

Definition 11 (Minimal Explanation)**.** A *minimal explanation* is a prime that contains the instance.

There can be different explanations that are consistent with the prediction that a CT-Net makes for a particular instance. From a user's perspective, simpler explanations are better, i.e. its length should be short, so that it can be easily comprehended. Also, shorter explanations usually cover more feature space, and thus provide users with more insights into the behavior of the CT-Net.

We now describe our algorithm for finding a minimal explanation for a positive prediction of a CT-Net. The pseudo code is outlined in Algorithm 3. In this algorithm, we start by treating the instance α itself as the current explanation π, and we then iteratively remove one feature at a time from the current explanation as long as a candidate feature can be identified such that the slack $z_j(\pi)$ remains non-negative for all threshold functions or until there are no more features.

Algorithm 3. Derive Minimal Explanation

```
Input: A set of threshold functions $F = \{f_1, f_2, \ldots, f_k\}$, positive instance $\alpha$
Output: A minimal explanation $\pi$
 1: $\pi \leftarrow \alpha$
 2: while $\pi \neq \emptyset$ do
 3:     $\mathscr{L} = \emptyset$
 4:     for $\ell_i \in \pi$ do
 5:         if $z_F(\pi \setminus \{\ell_i\}) \geq 0$ then
 6:             $\mathscr{L} = \mathscr{L} \cup \{\ell_i\}$
 7:         end if
 8:     end for
 9:     if $\mathscr{L} = \emptyset$ then
10:         return $\pi$
11:     else
12:         $\ell_i \leftarrow$ select_candidate($\mathscr{L}$)
13:         $\pi \leftarrow \pi \setminus \{\ell_i\}$
14:     end if
15: end while
16: return $\pi$
```

There are two key parts to Algorithm 3. The first key part is in Lines 3-8, in which a list $\mathscr{L}$ of features are identified as candidates for removal. A feature ℓ_i is a candidate for removal if its removal does not cause the slack of any threshold function to become non-negative. That is

$$z_j(\pi \setminus \{\ell_i\}) \geq 0, \forall j.$$

If no such candidate exists or if there are no more features, then we have arrived at a minimal explanation.

The second key part is in Line 12, in which the **select_candidate** function is called to select a candidate feature from the set $\mathscr{L}$. One approach is to select the feature ℓ_i from $\mathscr{L}$ that maximizes the *average slack* among the threshold functions. That is, let $z_j(\pi \setminus \{\ell_i\})$ be the slack for threshold function f_j by removing ℓ_i, and let

$$z_{avg} = \text{average}\{z_j(\pi \setminus \{\ell_i\})\}. \tag{3.8}$$

We then select the ℓ_i that maximizes z_{avg}.

Alternatively, we can select the feature ℓ_i from $\mathscr{L}$ that maximizes the *minimum slack* among the threshold functions. That is, let

$$z_{min} = z_F(\pi \setminus \{\ell_i\}) = \min\{z_j(\pi \setminus \{\ell_i\})\}. \tag{3.9}$$

In this alternative approach, the ℓ_i that maximizes z_{min} would be selected.

Experimentally, we found that maximizing the average slack (i.e., Equation 3.8) to be more effective, and thus this is the approach that we adopted in our evaluation section (*cf.* Section 3.6).

Theorem 8. *The explanation derived using Algorithm 3 is minimal.*

Proof. We prove this theorem by contradiction. Assume the explanation π_1 generated with Algorithm 3 is not minimal. Then there must exists another prime π_2 such that $\pi_1 \Rightarrow \pi_2$ (π_2 covers π_1). This implies there exists a literal ℓ_i such that $\ell_i \in \pi_1$ and $\ell_i \notin \pi_2$. Since π_2 is a prime, it must guarantee that all threshold functions in the hidden layer evaluate to true, i.e., $z_F(\pi_2) \geq 0$. This would mean further removing ℓ_i from π_1 would still produce a positive group slack. However, this is contradictory to Algorithm 3 since removing any additional feature from π_1 would lead to a negative group slack (i.e., $\mathscr{L} = \emptyset$). □

3.5 Sparsity-Promoting Training of CT-Net

We leverage the stochastic gradient descent (SGD) algorithm to efficiently train CT-Net. In particular, a binary cross-entropy function is used as the loss function to measure the error between the predicted output and the real labels. We recognize that the step functions have zero-gradients everywhere except 0, and we tackle this problem by employing a straight-through estimator approach [5] with a gradient clipping technique to back-propagate gradient updates through the activations of threshold functions in the hidden layer.

It should be clear from the previous section that zero weights in a threshold function mean that the corresponding inputs will not have any effect on the logic of that threshold function, and the corresponding threshold formula becomes simpler. Intuitively, maximizing the sparsity of the threshold functions in the hidden layer encourages simpler explanations. Further, our algorithms for deriving explanations can also benefit from having weights that have small absolute values. This is because they will less impact on the available slack of the corresponding threshold function when removed.

To incorporate the idea proposed above in the training process, we propose to add a sparsity-promoting regularizer to encourage both zero weights and weights with small absolute values. In particular, we employ an improved version of L_1 regularization called reweighted L_1 regularization [13], which drives the weights with smaller absolute values down to zero faster by giving those weights relative larger gradients. Mathematically, the reweighted L_1 minimization can be achieved by employing a log-sum penalty term as the regularization loss

$$\mathscr{L}_R(W) = \log(\|W\|_1 + \varepsilon), \tag{3.10}$$

where $\varepsilon > 0$ is a small value added to ensure numerical stability (e.g., $\varepsilon = 0.1$). To even encourage more zero weights in the threshold functions, we further prune the weights with absolute values below a certain threshold by setting them directly to zero [31].

As discussed in the previous section, zero weights in the output layer are also helpful in excluding the unnecessary threshold functions of the hidden layer from the conjunction. Therefore, the reweighted L_1 minimization is applied again to the output layer to promote sparsity. However, since the binarized weights are required for the output layer, along with pruning the weights below a certain threshold as in the hidden layer, during the feed-forward phase, we also set the weights above the threshold directly to one, while we maintain and keep updating the full-precision values of the weights through back-propagation. Note that negative weights are not expected in the output layer, so we initialize the full-precision weights to be all one and always prune the weights below the positive threshold (whereas weights are also compared with a negative threshold in the hidden layer).

With the regularizer applied to the hidden layer and output layer, the overall loss function we optimize for becomes as follows: we add a regularization term into the loss function of the form

$$\mathscr{L} = \mathscr{L}_{BCE} + \lambda_1 \mathscr{L}_{R_1} + \lambda_2 \mathscr{L}_{R_2}, \tag{3.11}$$

where $\mathscr{L}_{BCE}$ is the binary cross-entropy loss, $\mathscr{L}_{R_1}$ and $\mathscr{L}_{R_2}$ are the regularization loss as explained in Equation 3.10 for the hidden layer and output layer, respectively, and λ_1 and λ_2 are their corresponding regularization coefficients.

3.6 Experimental Evaluation

Datasets. In this section, we evaluated the proposed CT-Net along with a set of baseline approaches on 11 publicly available tabular classification datasets. Three datasets are from UCI Machine Learning Repository [27]: Adult Census (adult), MAGIC Gamma Telescope (magic), and Chess: King-Rook vs. King (chess). Four datasets are from Kaggle: Telco Customer Churn (churn), Churn Modelling (churn2), Dataset Surgical binary classification (surgical), and Airline Passenger Satisfaction (airline). The other four datasets are: House_16H (house) [89], TED Dataset (retention) [3], Predicting Recidivism in North Carolina, 1978 and 1980 (recidivism) [79] and Home Equity Line of Credit Dataset (heloc) [14]. Most of these datasets consist of more than 10,000 instances that originally include binary, categorical, and numerical attributes. More details of the datasets are shown in Table 3.2 As we can see from the "imbalance ratio" column of Table 3.2, the datasets selected vary from nearly balanced (1.09) to considerably imbalanced (3.91). Although other evaluation metrics such as F1-score might be better suited for imbalanced datasets, we choose to use the test accuracy that is much easier to interpret and widely used in the experiments of papers[25, 91] on the similar topic.

Table 3.2. Details of the datasets used in the experiment. The imbalance ratio is calculated as the number of negative instances divided by the number of positive instance.

dataset	# instances	# features	imbalance ratio
adult	30162	14	3.02
magic	19020	10	1.84
house	22784	16	0.42
recidivism	8680	16	1.72
chess	28056	6	1.48
retention	10000	8	1.96
churn	7032	19	2.76
airline	25893	22	1.28
heloc	10459	23	1.09
churn2	10000	10	3.91
surgical	14635	24	2.97

Baselines and Pre-processing. The baseline approaches evaluated as comparisons consist of four rule learners, including Decision Rule Net (DR-Net) [71], the Column-Generation-Based algorithm (CG) [25], RIPPER [20], and Bayesian Rule Sets (BRS) [91]; and three traditional machine learning classifiers, including decision trees (CART), random forests (RF), and gradient boosting trees (XGB). In particular, RIPPER is an old variant of the Sequential Covering algorithm for greedily mining rule set from the dataset, whereas DR-Net, BRS and CG are more recent rule-set-generation classifiers that optimize interpretability and accuracy at the same time. We use the CART [9] algorithm for learning decision trees, whereas random forest (RF) [10] and XGBoost (XGB) [15] serve as uninterpretable baselines to illustrate the typical performances that black-box models can achieve on the evaluated datasets. We used scikit-learn [69] implementations for CART and RF. The implementations of all baseline models are publicly available on GitHub [4]. For all datasets, we encoded the categorical and numerical attributes into binarized features according to the scheme described in [71]. Moreover, for BRS

[4] DR-Net (https://github.com/Joeyonng/decision-rules-network); CG (https://github.com/Trusted-AI/AIX360); RIPPER (https://github.com/imoscovitz/wittgenstein); BRS (https://github.com/wangtongada/BOA); scikit-learn (https://github.com/scikit-learn/scikit-learn); xgboost (https://github.com/dmlc/xgboost).

and CG, negations of the binarized features are appended along with their positive counterparts, e.g., non-smoker vs. smoker, according to the steps described in their experimental sections so that negative literals can be considered in their rule sets, which is required in their papers.

Table 3.3. Average test accuracy based on the nested 5-fold cross-validation. Standard deviations are in parentheses.

dataset	CT-Net	DR-Net	CG	RIPPER	BRS	CART	RF	XGB
adult	**84.08** (0.46)	82.55 (0.61)	82.60 (0.62)	82.25 (0.85)	78.78 (0.58)	82.44 (0.35)	84.03 (0.55)	84.41 (0.19)
magic	**84.91** (0.56)	83.91 (0.53)	83.33 (0.59)	82.86 (0.52)	81.37 (0.73)	83.18 (0.44)	86.71 (0.48)	87.16 (0.36)
house	**88.47** (0.40)	86.07 (0.41)	83.80 (0.78)	81.43 (3.19)	83.26 (0.55)	85.10 (0.60)	88.49 (0.19)	88.92 (0.35)
recidivism	**66.24** (1.00)	64.09 (0.46)	64.57 (0.67)	64.84 (0.36)	61.98 (0.75)	62.85 (0.87)	66.77 (0.66)	64.33 (1.25)
chess	**91.47** (0.42)	84.47 (0.51)	81.93 (0.50)	85.46 (1.04)	74.66 (2.15)	85.36 (0.40)	92.63 (0.42)	94.98 (0.36)
retention	**93.85** (0.46)	87.78 (0.37)	90.77 (0.57)	88.92 (0.58)	89.37 (1.60)	90.11 (0.65)	93.43 (0.42)	94.29 (0.28)
churn	**80.36** (1.30)	78.85 (0.61)	79.21 (1.07)	78.27 (0.39)	76.74 (1.28)	79.00 (0.57)	80.35 (0.93)	77.45 (0.96)
airline	**95.03** (0.41)	93.32 (0.30)	90.10 (0.31)	93.08 (1.27)	90.71 (0.46)	90.21 (0.43)	94.79 (0.42)	95.94 (0.23)
heloc	**71.69** (0.80)	71.36 (0.75)	70.05 (0.43)	68.85 (1.19)	70.82 (0.74)	70.00 (1.19)	71.95 (0.67)	70.39 (0.56)
churn2	85.35 (0.34)	**85.97** (0.07)	85.68 (0.59)	85.07 (0.39)	85.89 (0.55)	84.33 (0.08)	86.05 (0.54)	85.26 (0.68)
surgical	83.64 (0.75)	**84.78** (0.58)	80.38 (0.58)	83.35 (1.05)	80.25 (0.49)	79.40 (0.59)	82.90 (0.31)	85.26 (0.33)

CT-Net Configurations and Parameter Tuning. We used the Adam optimizer with a fixed learning rate of 10^{-3} when evaluating CT-Net. In addition, we incorporated the sparsity-promoting regularization discussed before, so we did not further apply L_2 regularization (weight decay) to the experiments. The neural networks were constructed with 100 neurons in the hidden layer and we let our regularization technique prune the unnecessary neurons. For simplicity, we used a mini-batch size of 2,000 and all networks were trained for 2,000 epochs to guarantee complete convergence. We employed the nested 5-fold cross-validation to select the parameters that maximize the training accuracy. In particular, each dataset was shuffled (with a fixed seed to ensure the consistency for all approaches) and split into 5 training-testing pairs, for each of which we derived a set of parameters that maximize the accuracy on the training subset. The parameters were then adopted to evaluate the corresponding training-testing pair and the final results were produced by averaging the performance over the 5 pairs. To be specific, we tune the regularization coefficients λ_1 and λ_2 for CT-Net, the minimum number of samples per leaf for CART and RF, the regularization term for XGBoost, and the same parameters for DR-Net, CG, RIPPER, and BRS as discussed in [71].

Classification Performance. The classification performances of CT-Net and other baseline approaches were evaluated based on accuracy. In particular, the accuracy is the test accuracy computed based on the nested 5-fold cross-validation as previously explained. The accuracies for all models are summarized in Table 3.3. As can be observed in Table 3.3, CT-Net achieves the best test accuracy among all interpretable models across all datasets except for churn2 and surgical, which is close to or even higher than the black-box classifiers on some of the datasets (e.g., adult, recidivism, and retention). This reflects the significant advantage of CT-Net on its generalization capability.

We further analyzed the results in Table 3.3 using a two-step procedure recommended in [26], which consists of a Friedman test to check whether all classifiers perform similarly and a follow-up Nemenyi test to compare pairs of the classifiers. Applying the Friedman test, we derived Friedman statistic to be 40.39, which is larger than the critical value of the chi-squared distribution with 7 degrees of freedom $\chi^2_7 = 14.07$ for $\alpha = 0.05$. Thus we can reject the null hypothesis that all classifiers tested have the equal performance. Then we continue to use Nemenyi test to compare whether there is significant difference between all pairs of the classifiers and the results are shown in Figure 3.2. As we can see from the figure, CT-Net has the second

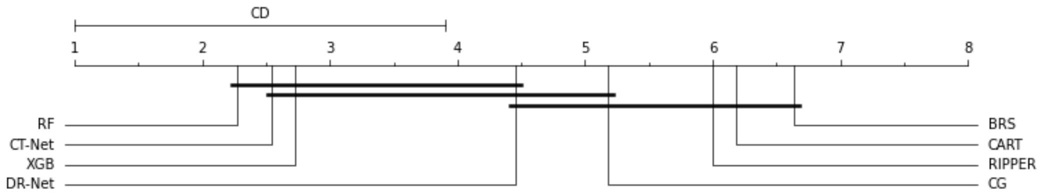


Figure 3.2. Results of Nemenyi test for all classifiers. Groups of classifiers that are not significantly different at $\alpha = 0.1$ are connected.

highest average rank that is positioned between Random Forest and XGBoost, proving that CT-Net can achieve the state-of-the-art predictive performances similar to other black-box models. Compared with the interpretable models, CT-Net has shown to be statistically significantly better than RIPPER, CART, and BRS in terms of the testing accuracy, which demonstrates that CT-Net can be considered as a great alternative to other interpretable models.

Table 3.4. Statistics of CT-Net after pruning, where # pruned is the average number of neurons pruned in the hidden layer and sparsity represents the average percentage of the zero weights in the remaining neurons.

dataset	# pruned	sparsity
adult	97.0	63.28%
magic	96.8	47.00%
house	97.2	54.21%
recidivism	79.2	44.75%
chess	45.4	72.57%
retention	72.8	71.83%
churn	97.2	71.20%
airline	89.6	65.30%
heloc	98.2	43.05%
churn2	77.2	79.59%
surgical	96.8	67.93%

Effects of Sparsity-Promoting Regularization. In our evaluation, the number of threshold functions, i.e., the number of effective neurons in the hidden layer, can be up to 100 depending on the parameter tuning, which at the same time guarantees enough generalization capacity and increases the complexity. As discussed in Section 3.5, we employed the sparsity-promoting regularization technique in our training procedure to attenuate overfitting and simplify the threshold functions. In particular, our regularization approach can not only sparsify the hidden layer, but also directly turn off the entire threshold function by pruning its corresponding weight in the output layer. We evaluated the performance of our regularization technique in both ways and the results are summarized in Table 3.4. As can be observed, the number of pruned neurons varies significantly from 45% (chess) to 98% (heloc), which validates the effectiveness of our regularization approach in removing redundant capacity while preserving the necessary neurons based on the complexity of the dataset. Further, the average sparsity of the remaining neurons is generally greater than 50% except for the magic, recidivism, and heloc datasets, indicating the training procedure effectively excludes any literals that have little contribution to the prediction. In particular, it can be seen that while a relatively small number of neurons are pruned for chess, the remaining sparsity is higher than other datasets. This suggests that regularization can capture the underlying logic of the datasets and find a correct direction to simplify the neural network without severely hurting the generalization.

3.7 Conclusion

In this paper, we proposed a three-layer neural network architecture called CT-Net that can be trained for classifying tabular data to achieve high prediction accuracy. In particular, the trainable hidden layer neurons with step activation function logically correspond to a set of threshold logic functions, while the output layer further constructs a conjunction of these threshold functions. In addition, once the network is trained, for any positive prediction, a provably minimal explanation can be readily derived from the model. We further adopt a sparsity-promoting regularization technique to sparsify the network and simplify the threshold functions. Experimental results demonstrate that our approach has significant advantages on producing accuracy predictions over other state-of-the-art interpretable decision models.

Several potential improvements can be developed in the future work. First, while our proposed output layer essentially performs a logical conjunction (AND), other logic operations can also be used in place of the conjunctive operation, including OR, XOR, or simply a standard fully-connected layer. Second, multiple proposed layers that encode a logical operation can be stacked together to provide higher network capacities. Third, the current work is focused on binary classification with a single output neuron. Future work may extend the network to be a multi-class classifier by increasing the number of output neurons with additional modifications.

3.8 Acknowledgements

This research has been supported in part by the National Science Foundation (NSF IIS-1956339).

3.9 Acknowledgements

Chapter 3, in full, is a reprint of the material as it appears in Machine Learning with Applications, 2022. Weijia Wang, Litao Qiao, and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Alternative Formulations of Decision Rule Learning from Neural Networks

4.1 Introduction

Deploying inherently interpretable decision models that can provide human understandable explanations is critically important in machine learning domains like healthcare and criminal justice where human lives are often deeply impacted [77]. In these domains, the datasets are typically provided as tabular data with naturally meaningful features. One popular approach to tabular learning is the use of decision rule sets [20, 86, 47, 91, 25]. In decision rule sets, the model is represented in disjunctive normal form (DNF) as an independent set of logical rules.

Decision rules are inherently interpretable: the rules are expressed in terms of logical combinations of input conditions that must be satisfied for a positive prediction. In addition to providing a prediction, the corresponding matching rule in the model also serves as an explanation that humans can easily understand. In particular, the explanations are stated directly in terms of meaningful input features, which can be categorical (e.g., color equal to red, blue, or green) or numerical (e.g., score $\geq$ 100) attributes, where the binary encoding of categorical and numerical attributes is well-studied [91, 25]. Although decision rules can provide explainable predictions, they often produce inferior results in terms of accuracy when compared to models like gradient boosted and ensemble decision trees [15, 10]. However, these black-box models are difficult or impossible for humans to understand. Their lack of interpretability makes them difficult to gain public trust for their use in high-stakes applications like medical-diagnosis and criminal justice, where decisions can have serious consequences on human lives [77].

Recently, [71] proposed a new paradigm for decision rule learning as a neural net training problem in which the proposed DR-Net neural network architecture maps directly to an AND-OR logic network in disjunctive normal form (DNF), which can be readily translated into an explainable decision rule set for binary classification. In particular, DR-Net is a three layer architecture in which the first layer comprises the inputs, the hidden layer comprises neurons that implement trainable logic-AND operators, and the output neuron implements a trainable logical-OR operation. The trainable logical-AND neurons form rules as a conjunction of input features, and the trainable logical-OR neuron forms a rule set as a disjunction of rules. As detailed in [71], this approach can leverage a large body of sophisticated neural net training techniques to achieve state-of-the-art predictive performance while retaining the interpretability of decision rules.

Despite the advances made in the DR-Net work, there are several issues that deserve further attention, which we aim to address in this paper. One issue is that the AND neurons and the OR neurons in [71] are formulated differently: the trainable AND neurons have *continuous* weights, whereas the trainable OR neurons require the *binarization* of weights to 0 or 1. The different treatments of the two logic operators make it more difficult to compose them together in future multi-level neural net architectures. In addition, their different treatments also make it more difficult to employ certain sparsity-promoting neural net training techniques like reweighted L_1 regularization, which are important for achieving sparse networks that translate to simpler decision rules for tabular learning.

In this paper, we have substantially extended the work in [71] in the following ways:

- We propose a new formulation of a trainable OR neuron based on continuously trainable weights without the need to binarize the weights, in the same way that the AND neuron is formulated. Moreover, our new formulation of the OR neuron is generalized in the same way as our AND neuron formulation in that inputs can now be negated by means of negative weights. This creates more flexibility in the training process.

- We further added a new trainable "NAND neuron" (Not AND), which is also based on the same model of trainable continuous weights.

- By De Morgan's Law, we know that an AND-OR logic network is logically equivalent to a NAND-NAND logic network. Therefore, given our formulation of a trainable NAND neuron, we propose a new neural net architecture called NN-Net (short for NAND-NAND Net) for decision rule learning as an alternative to DR-Net. We also modified the formulation of DR-Net with our new formulations of the AND and OR neurons.

- Further, given our new formulations of Boolean logic operators as trainable neurons with continuous weights, existing sparsity-promoting neural net training techniques like reweighted L_1 regularization can be directly applied to derive simpler decision rule sets, in addition to a stochastic L_0 regularization approach that was previously used in [71].

- In addition, we added many new experiments in our experimental evaluation section. In particular, we evaluated our new formulations of DR-Net and NN-Net, together with two sparsity-promoting regularization approaches. We also added new experiments to analyze the training process and show the effects of the proposed sparsity-promoting mechanisms, including the analysis of training convergence and the effects of using different combinations of regularization coefficients, which affect the model complexities.

The rest of the paper is outlined as follows: Section 4.2 provides some background. Section 4.3 systematically defines different logic operators as fundamental building blocks of networks. Section 4.4 introduces a new version of DR-Net and proposes NAND-NAND net as an alternative network structure that can also be mapped to a set of decision rules. Section 4.5 describes sparsity-promoting regularization approaches for training the proposed networks.

Section 4.6 provides extensive evaluations of our proposed approaches. Section 4.7 summarizes related work. Section 4.8 concludes the paper.

4.2 Background

4.2.1 Binarization of tabular data

Although binary features commonly appear in tabular datasets, these datasets also generally include categorical and numerical features, which are naturally used when the data is collected. In this work, we assume all data are binary encoded and thus categorical and numerical features need to be first binarized using well established preprocessing steps in the machine learning literature. In particular, we follow exactly the same binarization approach used in some decision ruler learners [91, 25], where we simply one-hot encode all categorical features into binary vectors. For numerical features, we adopt quantile discretization based on the distribution of numerical values in the training data to get a set of thresholds for each feature, where the original numerical value is one-hot-encoded into a binary vector by comparing with the thresholds (e.g., age $\leq$ 25, age $\leq$ 50, age $\leq$ 75) and encoded as 1 if less than the threshold or 0 otherwise. This binarization approach for numerical features has been widely used by decision rule learners and shown to achieve better performance than directly discretizing numerical values into intervals [91].

4.2.2 The decision rule learning problem

Once a tabular dataset has been binarized, as explained above, the goal of decision rule learning is to train a classifier in the form of a Boolean logic function in disjunctive normal form (OR-of-ANDs). Each logical-AND operation serves as a decision rule by forming the conjunction of a subset of input features or their negations. An instance satisfies a rule if all the conditions captured in the rule are satisfied for the instance. The logical-OR operation serves to form a decision rule set by forming the disjunction of the rules. The logical-OR operation means the final prediction will be positive if at least one rule is satisfied (i.e., the AND operation is true). Otherwise, the final prediction will be negative.

In particular, a training set can be represented mathematically as a set of N data samples $(\mathbf{x}_n, y_n)$, $n = 1, ..., N$, where $\mathbf{x}_n$ is a vector of D binarized features $x_{n,i} \in \{0,1\}$, $i = 1, ..., D$, and $y_n \in \{0,1\}$. The decision rule set learned can be denoted as a set of m terms: $C = \{c_1, c_2, ..., c_m\}$. We define a term c to be a conjunction of k features (e.g., an input feature x_i) or their negations (e.g., $\bar{x}_i$), where $1 \leq k \leq D$. If an input feature x_i or its negation $\bar{x}_i$ are both excluded from the term c, we say x_i is a "don't care," meaning whether x_i is 0 or 1 has no effect on the outcome of term c. Under this definition, an instance $\mathbf{x}_n$ satisfies a term if only if all conditions in the terms are satisfied in the instance: i.e. $x_{n,i} = 1$ for x_i and $x_{n,i} = 0$ for $\bar{x}_i$.

4.3 Boolean Logic Operators as Trainable Neurons

In this section, we present the formulation of several Boolean logic operators as trainable neurons. In particular, we describe the formulation of three trainable logic operators, AND, OR, and NAND (Not AND) that will be used as building blocks in the neural net architectures described in the next section. Unlike the earlier work in [71] that treated the AND and OR neurons differently, we formulate all three logic operators in the same way in this paper as trainable neurons with continuous weights and dynamic biases. We believe this uniform formulation of all three logical neurons is cleaner and enables a more consistent training of the neural nets that use them.

4.3.1 AND Neuron

As discussed in Section 4.2.1, categorical and numerical attributes are first binarized into Boolean vectors. Therefore, decision rules become simply a conjunction (or a logical-AND) of the corresponding binary variables. We would like to define a neuron that is *trainable* in the following sense: Given $\mathbf{x} \in \{0,1\}^D$ as a vector of D binary variables, we want to define a neuron that can be trained to implement the conjunction of a *subset* of these D variables, depending if the corresponding weights are non-zero or not. Zero weights would be interpreted as the *exclusion* of the corresponding binary variables in the conjunction subset. Further, for the binary variables included in the conjunction subset, we would like to *generalize* the AND operator to include the conjunction of a binary variable x_i or its negation $\bar{x}_i$, depending if the corresponding weight w_i is positive or negative. This is achieved by defining a *soft* formula, followed by a binary step activation function.

Specifically, given the binarized inputs as $\mathbf{x} \in \{0,1\}^D$ and the output as y, a neuron that performs a *soft AND* operation is defined as follows:

$$y_{\text{AND}} = \sum_{i=0}^{D} w_i x_i - \sum_{w_i > 0} w_i. \tag{4.1}$$

In Equation 4.1, the dot product of the weights and inputs ($\sum_{i=0}^{D} w_i x_i$) is added with a *dynamic bias* term ($-\sum_{w_i>0} w_i$), whose value depends on all the positive weights of the neuron. With the dynamic bias and binarized inputs, the range of the output of a soft AND neuron is within $(-\infty, 0]$. The output $y = 0$ can only be achieved when all inputs *match* the sign of the corresponding weights: all positive weights should have the inputs of 1 and all negative weights should have the inputs of 0. Thus, a soft AND neuron can be seen as a logic AND gate if we map the neuron output of 0 as TRUE and other negative neuron outputs as FALSE. Just like the behavior of weights in regular neurons, the zero weights in a soft AND neuron mean that the corresponding inputs will not have any effect on the output.

However, the outputs of the soft AND neurons cannot be directly passed as inputs to other soft AND neurons because they are not binary numbers, and thus multiple layers of soft neurons alone cannot be concatenated to form a neural network that performs logical operations. Thus, in order for soft AND neurons to function as proper logical gates in the neural network forward process, binary step functions are applied as the activation functions.

The binary step activation function for the soft AND neurons is defined as

$$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

which simply maps the value 0 to value 1 and negative values to value 0. The activation function defined in Equation 4.2 turns a soft AND neuron into a *hard* AND neuron that exactly performs a logical AND gate operation, since the output is 1 only when all of the input operands are TRUE (all inputs match the sign of the corresponding weights) and 0 otherwise. However, as can be observed, the binary step function defined in Equation 4.2 discretizes continuous inputs into binary integers, which is not naturally differentiable and the classic gradient computation approach doesn't apply here. Therefore, we utilize the straight-through estimator discussed in [5] with the gradient clipping technique. Denoted by $\hat{y}_i$ the binarized activation based on y_i, we compute the gradient as follows:

$$g_{\hat{y}_i} = \begin{cases} 0 & \text{if } y_i < -1 \\ g_{y_i} & \text{otherwise,} \end{cases} \tag{4.3}$$

where $g_{\hat{y}_i} = \frac{\partial L}{\partial \hat{y}_i}$ and $g_{y_i} = \frac{\partial L}{\partial y_i}$ are the gradients of classification loss L w.r.t. $\hat{y}_i$ and y_i, respectively. The condition $y_i < -1$ is motivated by the ReLU1 function, which clips the gradient w.r.t. the outputs that are more than 1 away from the maximum value and introduces non-linearity into the training process and empirically improves the performance.

4.3.2 OR Neuron

Like the trainable AND neuron described in the previous section, we would like to define an OR neuron that can be trained to implement the *disjunction* (or the logical-OR) of a *subset* of the input variables, with zero weights interpreted as the exclusion of the corresponding binary variables. Also, like our formulation of the AND neuron, we generalize the OR neuron to include the disjunction of a binary variable x_i or its negation $\overline{x}_i$, depending if the corresponding weight w_i is positive or negative.

In particular, similar to the soft AND neuron, we define a *soft OR* neuron as follows[1]:

$$y_{\text{OR}} = \sum_{i=0}^{D} w_i x_i - \sum_{w_i<0} w_i. \tag{4.4}$$

A soft OR neuron is similar to a soft AND neuron in that it also consists of the dot product and a dynamic bias, but the dynamic bias now depends on all negative weights of the neuron, as opposed to the positive weights in the soft AND neuron. Because of the flipping sign of the dynamic bias in the soft AND neuron, the range of the output is also flipped to be $[0, \infty)$, where 0 means FALSE and can only be produced when all inputs do *not* match the sign of the corresponding weights. Although 0 values can appear in the outputs of both soft AND neuron and soft OR neuron, the interpretations are quite contrasting: it represents TRUE for soft AND neurons but FALSE for soft OR neurons.

One of the benefits of using the formulations mentioned above to define soft AND and OR neurons is that the neurons are interchangeable with conjunctions and disjunctions while at the same time being fully differentiable. The dynamic bias provides logical meanings to the signs of trainable weights and values of the outputs. In particular, the operands of soft AND and OR operations are TRUE when the inputs to the neurons match the signs of the corresponding weights, and soft AND and OR neurons output 0 (considered TRUE for AND neuron and FALSE for OR neuron) only when *all* the operands are TRUE and FALSE respectively. Also, dynamic bias only involves linear operations such as multiplications and deductions and thus is fully differentiable, which helps the gradients flow smoothly in the backward propagation process.

[1] In [71], the OR neuron is formulated differently. The soft OR operation in that earlier work is defined as $y_{\text{OR}} = \sum_{j=1}^{D} \hat{w}_i x_i - \varepsilon$, where $0 < \varepsilon < 1$ is a small value (e.g., $\varepsilon = 0.5$), and where $\hat{w}_i$ is the *binarized* version of the full-precision weight w_i such that $\hat{w}_i = 0$ if $w_i \leq 0$ and 1 otherwise. This required *binarization* of weights is different from the formulation of the AND neuron that uses *continuous* weights. The different treatments of the two logic operators make it more difficult to compose them together in future multi-level neural net architectures, and their different treatments also make it more difficult to employ certain sparsity-promoting neural net training techniques like reweighted L_1 regularization that we use in this work.

The soft OR neuron can be extended to be a *hard* OR neuron by applying a similar binary step activation function

$$f(x) = \begin{cases} 1 & \text{if } x \neq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where the gradients are computed as

$$g_{\hat{y}_i} = \begin{cases} 0 & \text{if } y_i > 1 \\ g_{y_i} & \text{otherwise.} \end{cases} \tag{4.6}$$

As can be seen later, the hard logic neurons are necessary building blocks of Decision Rules Network, ensuring that the inferences of the network and the derived decision rules are identical.

4.3.3 NAND Neuron

Apart from the AND and OR operations, the NAND operator can also be used as a building block for the decision rule set. As discussed in the next section, the concatenation of two NAND operations is equivalent and can be easily transformed to the OR-of-AND form. Since the NAND operation is defined as an AND operation with the output negated, we can define the *soft NAND* neuron by appending a negation function to the output of the soft AND neuron:

$$y_{\text{NAND}} = -y_{\text{AND}} = \sum_{w_i > 0} w_i - \sum_{i=0}^{D} w_i x_i. \tag{4.7}$$

Remember that the soft AND neuron defined in Equation 4.1 will output a continuous value between $(-\infty, 0]$, where the output 0 can only be attained when all inputs match the sign of the corresponding weights. The negation function in Equation 4.7 simply inverted the soft AND neuron to output a value within the range $[0, \infty)$. Since the output of the soft NAND neuron can be interpreted in the same way as the output of the soft OR neuron, the binary step activation function for the soft OR neuron defined in Equation 4.5 can be used to turn the soft NAND neuron to the hard NAND neuron. Therefore, the soft NAND operation, which consists of a soft AND operation and a negation function, together with the binary step activation function defined in Equation 4.5 forms a logical NAND gate that outputs 0 when all inputs match the sign of the corresponding weights and 1 otherwise. To summarize, a soft NAND neuron is implemented using a soft AND neuron with a negation function and a hard NAND neuron is achieved by applying a step binary activation function afterward.

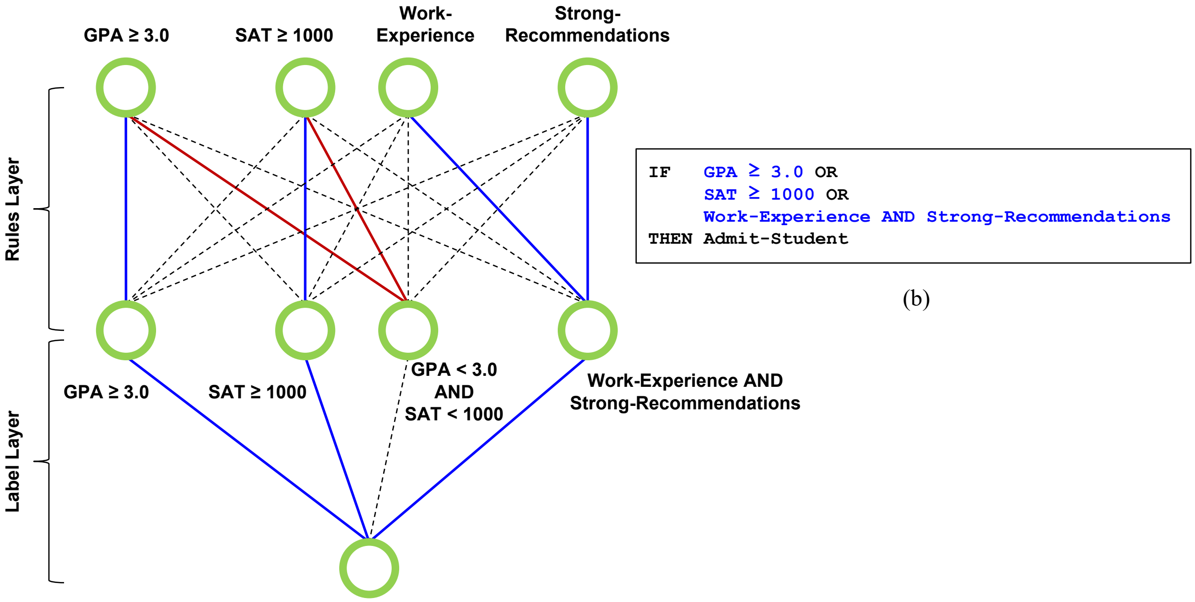
4.4 Design Rule Learning as Trainable Neural Networks

In this section, we first review the DR-Net architecture proposed in [71]. However, in this work, we replace the AND and OR operators with our new formulations in Section 4.3. In addition, we introduce an alternative neural network structure called a NAND-NAND net by leveraging the NAND neuron formulation, which also maps correspondingly to a set of decision rules in disjunctive normal form (DNF) in accordance to De Morgan's Law.

4.4.1 Decision Rules Network

As described in [71], the Decision Rules Network (DR-Net for short) is a simple-three layer neural network architecture, comprising an input layer of n input units, a hidden layer of k neurons, and an output layer with a single output neuron. A toy example is shown in Figure 4.1(a) for predicting college admissions, which we use to illustrate several key ideas regarding the DR-Net architecture. In this example, the first input "GPA $\geq$ 3.0" indicates that the student has at least a high school grade point average (GPA) of 3.0. The second input "SAT $\geq$ 1000 indicates that the student has scored at least 1000 on the SAT college entrance exam. The last two inputs indicate whether or not the student has work experience and strong letters of recommendation, respectively.

Each of the n units at the input layer passes its corresponding assigned binarized value to each neuron in the hidden layer. Denoted as Rules Layer, the hidden layer contains k AND neurons to implement k logical-AND operations. The output unit, denoted as Label Layer,



```
IF    GPA ≥ 3.0 OR
      SAT ≥ 1000 OR
      Work-Experience AND Strong-Recommendations
THEN  Admit-Student
```

(b)

(a)

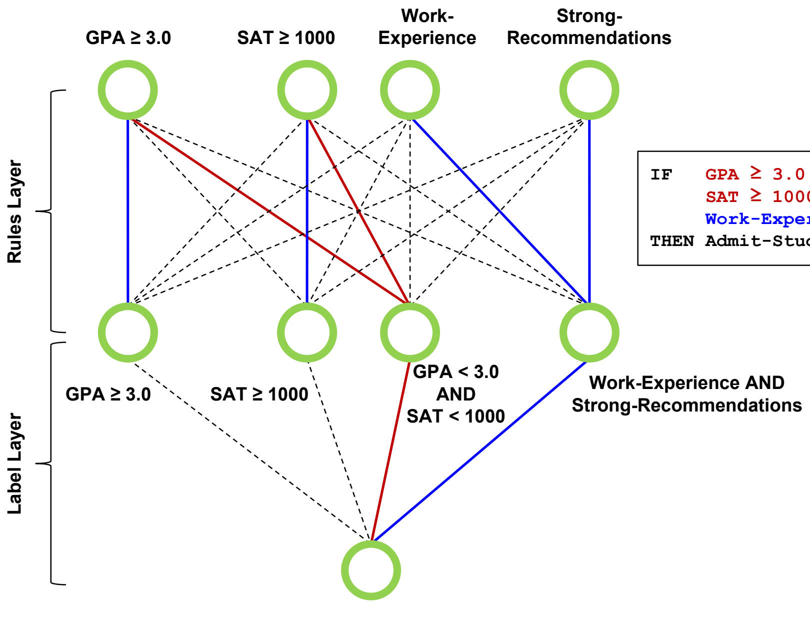
Figure 4.1. (a) An example of the DR-Net architecture with 4 AND neurons. The blue lines to the AND neurons represent positive weights while red lines represent negative weights. A dashed line indicates the exclusion of the corresponding input feature. Please note that we represent "NOT (GPA $\geq$ 3.0)" as "GPA $<$ 3.0" in the third rule. Similarly, "NOT (SAT $\geq$ 1000)" is represented as "SAT $<$ 1000." For the output OR neuron, a blue line indicates that the corresponding rule is included in the rule set, and a dashed line indicates the corresponding rule is excluded. (b) The network maps directly to the corresponding decision rule set shown in the box on the right.

implements a *disjunction* (logical OR) of the k AND neurons in the hidden layer.

In the Rules Layer shown in Figure 4.1(a), the blue lines indicate positive weights, the red lines indicate negative weights, and the dashed lines indicate zero weights, which means the corresponding inputs are excluded from the rule formed. For example, the first hidden layer neuron is interpreted as the rule "GPA $\geq$ 3.0." For the third hidden layer neuron, the red lines indicate "NOT (GPA $\geq$ 3.0) AND NOT (SAT $\geq$ 1000)," which is equivalently represented as the rule "GPA $<$ 3.0 AND SAT $<$ 1000." The last neuron is interpreted as the rule "Work-Experience AND Strong-Recommendations." Similarly, in the Layer Layer, the blue lines again indicate positive weights, which means the corresponding rule is included in the rule set, and the dashed lines indicate zero weights, which means the corresponding rule is excluded from the rule set. Together with the hidden layer of AND neurons, the overall DR-Net architecture implements a trainable AND-OR network as a Boolean formula in DNF. which directly maps to an unordered set of IF-THEN rules, as shown for example on the right-hand side of Figure 4.1(b).

The key difference between this work and our earlier work in [71] is in the definitions of the AND and OR neurons, as described in Section 4.3, in that both AND and OR neurons are uniformly formulated the same way with continuous weights, which enables them to be trainable with different regularization methods, for example the sparsity-based regularization methods described in Section 4.5. In addition, our OR neuron formulation is generalized to allow for the *negation* of inputs. As explained in Section 4.3, the OR neuron is generally defined so that it can include the disjunction of a binary variable x_i or its negation $\bar{x}_i$, depending if the corresponding weight w_i is positive or negative.

A variant of Figure 4.1 is shown in Figure 4.2. As shown in Figure 4.2(a), a negative weight is indicated by a red line in the Label Layer, which corresponds to the *negation* of the corresponding rule. In Figure 4.2(a), the AND rule for the third hidden layer neuron is "GPA $<$ 3.0 AND SAT $<$ 1000." The corresponding red line to the OR output neuron means the negation of this rule. By De Morgan's Law, this negation simply rewrites the AND rule by negating each binarized feature and OR-ing them together, which means the result logic will again be



(a)

```
IF   GPA ≥ 3.0 OR
     SAT ≥ 1000 OR
     Work-Experience AND Strong-Recommendations
THEN Admit-Student
```

(b)

Figure 4.2. (a) A variation of the example in Figure 4.1 in which the red line to the output OR neuron indicates the negation of the corresponding rule "GPA $<$ 3.0 AND SAT $<$ 1000." By De Morgan's Law, the negation of "GPA $<$ 3.0 AND SAT $<$ 1000" becomes "GPA $\geq$ 3.0 OR SAT $\geq$ 1000," which results in the same decision rule set. (b) The corresponding decision rule set is shown on the right.

in disjunctive normal form. In particular, the negation of “GPA $<$ 3.0” is “GPA $\geq$ 3.0” and the negation of “SAT $<$ 1000” is simply “SAT $\geq$ 1000.” Therefore, the negation of “‘GPA $<$ 3.0 AND SAT $<$ 1000” is simply “GPA $\geq$ 3.0 OR SAT $\geq$ 1000,” which results in the same set of IF-THEN rules, as shown in Figure 4.2(b). The corresponding rule to the red line is also shown in red in the IF-THEN rules shown in Figure 4.2(b).

4.4.2 NAND-NAND Network

Besides modifying the DR-Net architecture with our formulations of AND and OR neurons, we also propose an alternative architecture of the Decision Rules Network called a NAND-NAND Network (NN-Net), which is also a three layer fully-connected neural network that translates to a decision rule set in the disjunctive normal form with the categorical and numerical attributes binarized based on the same strategy with DR-Net. In particular, the main difference between NN-Net and DR-Net is that while the Rules Layer and the Label Layer of DR-Net encode a set of logical AND operators and an OR operator, respectively, NN-Net implements two levels of NAND operators cascading together with its Rules and Label layers.

As previously discussed, the set of decision rules in disjunctive normal form derived from DR-Net can be viewed as OR-of-ANDs, which are naturally implemented with a set of AND operators connecting to an OR operator. Mathematically, denote by $p_{k,i} \in \{x_i, \bar{x}_i\}$ the i-th predicate in the k-th rule, a decision rule set is expressed as follows:

$$y = \bigvee_k \bigwedge_i p_{k,i}. \tag{4.8}$$

where $\bigvee$ and $\bigwedge$ represent an AND operation and an OR operation, respectively. According to De Morgan’s Laws, the negation of a disjunction is the conjunction of the negations: $\overline{\bigvee p} = \bigwedge \overline{p}$. As a result, the DNF in Equation 4.8 can be converted to two levels of conjunctions as follows:

$$\bigvee_k \bigwedge_i p_{k,i} = \overline{\overline{\bigvee_k \bigwedge_i p_{k,i}}} = \overline{\bigwedge_k \overline{\bigwedge_i p_{k,i}}}, \tag{4.9}$$

where $\overline{\bigwedge p}$ corresponds to a logical-NAND (NOT-AND) operation. In other words, NAND-of-NANDs is logically equivalent to OR-of-ANDs. Therefore, NN-Net has the exact same structure as DR-Net where the hidden layer contains n hard NAND neurons and the output layer only has 1 soft NAND neuron.

Although formulated differently, NN-Net can be interpreted in the same way as DR-Net. That is, the hard NAND neurons in the Rules Layer of NN-Net encode rules in the same ways as the hard AND neurons in the Rules Layer of DR-Net, which will then be selected by the soft NAND neuron in the Label layer. The equivalence of translation to the decision rule set between NN-Net and DR-Net can be explained by the following two observations: 1) negations of the outputs from the Rules Layer NAND neurons can be separated out and folded into the Label Layer and 2) the negations of the inputs to the NAND neuron in the Label layer is essentially performing an OR operation according to De Morgan's Laws. The first observation shows that a set of conjunction rules can be readily translated from the weights of the Rules layer in NN-Net just like how they are from DR-Net, while the second one reveals that the way of interpreting the disjunction of the conjunction rules derived from the Rules layer is also through looking at the weights of the output neuron of NN-Net. More specifically, in light of the extra negation of the inputs borrowed from the hidden layer, the NOT-NAND neuron in the Label Layer of NN-Net will output true if one of the neurons in the Label Layer has the output that matches the sign of the corresponding weight. Thus, the neuron in the Label Layer combines the rules encoded in the Rules layer disjunctively, where the positive and negative weights require the positive and negative associations of the corresponding rules respectively, while the zero weights discard the rules completely.

4.5 Simplifying Rules through Sparsity

The neural network structures proposed above outline a way to derive a set of decision rules using stochastic gradient descent. As discussed above, for both DR-Net and NN-Net, a zero weight in a Rules layer neuron corresponds to the exclusion of the corresponding input feature, and a zero weight for the Label Layer neuron corresponds to the exclusion of the corresponding rule from the rule set. Thus, maximizing the *sparsity* of the Rules Layer corresponds to reducing the number of conditions in the rule set, and maximizing the sparsity of the Label Layer corresponds to eliminating more rules in the rule set. However, to eliminate an input feature from a logical rule or a logical rule from the complete rule set, the corresponding weight has to be exactly zero, which is difficult to achieve in the typical network training process.

To explicitly reduce the complexity of the derived rule set, the earlier work [71] only proposed one sparsity-promoting mechanism that leverages the recently proposed L_0 regularization. In this work, the new uniform definitions of AND, OR, and NAND neurons enable more versatile punning methods to be applied in our networks without any special adaption. To achieve a high degree of sparsity with exact zero weights, we have experimented with incorporating two sparsity-based mechanisms into the training process: 1) the recently proposed L_0 regularization that introduces trainable mask variables that are attached to all weights, and 2) the reweighted L_1 regularization [13] approach that drives the weights with small absolute values to zero by employing a log-sum penalty term.

In particular, the sparsity-promoting regularization is applied to both the Rules Layer and the Label Layer as follows:

$$\mathscr{L}_R = \lambda_1 \mathscr{L}_{R_1} + \lambda_2 \mathscr{L}_{R_2}, \tag{4.10}$$

where $\mathscr{L}_{R_1}$ and $\mathscr{L}_{R_2}$ are the regularization penalty terms w.r.t. the Rules Layer and the Label Layer, respectively, and λ_1 and λ_2 are the corresponding regularization coefficients that balance the classification accuracy and the rule set complexity. Intuitively, larger λ_1 and λ_2 will result in fewer number of conditions per rule and fewer number of rules, respectively. With the regularization techniques formulated by Equation 4.10 incorporated, the overall loss function we optimize for can be expressed as follows:

$$\mathscr{L} = \mathscr{L}_{\text{BCE}} + \mathscr{L}_R, \tag{4.11}$$

where $\mathscr{L}_{\text{BCE}}$ is the binary cross-entropy loss.

4.5.1 Sparsification with L_0 regularization

As discussed in [58], to achieve network sparsity through L_0 regularization, a binary random variable $z_i \in \{0, 1\}$ is attached to each weight of the model to indicate whether the corresponding weight is kept or removed. Therefore, each weight w_i can be represented by the product of a weight $\tilde{w}_i$ and the corresponding binary random variable z_i:

$$w_i = \tilde{w}_i z_i. \tag{4.12}$$

Assuming each z_i is subject to a Bernoulli distribution with parameter π_i, i.e., $q(z_i|\pi_i) = \text{Bern}(\pi_i)$, the probability that z_i is 1 is just π_i. In [58], L_0 regularization is implemented by summing all π_i parameters as the regularization term in the loss function, which penalizes the probabilities of masks being 1 and thus increases the sparsity of the network. Applying the above regularization method to both DR-Net and NN-Net is straightforward: all weight parameters are replaced by their product with the corresponding mask variables.

We denote by $\pi_{1,i,j}$ and $\pi_{2,j}$ the penalty of the non-zero mask variables of the neurons in Rules Layer and the output neuron in the Label Layer, respectively, where $i = 1, 2, \ldots, D$ is the feature index, and $j = 1, 2, \ldots, m$ is the index to the j-th neuron. Then the regularization loss $\mathscr{L}_R$

in Equations 4.10 and 4.11 can be specified as follows:

$$\begin{aligned}\mathscr{L}_R &= \lambda_1 \mathscr{L}_{R_1} + \lambda_2 \mathscr{L}_{R_2} \\ &= \lambda_1 \sum_{j=1}^{m} \sum_{i=1}^{D} \pi_{1,i,j} + \lambda_2 \sum_{j=1}^{m} \pi_{2,j}.\end{aligned} \tag{4.13}$$

4.5.2 Sparsification with reweighted L_1 regularization

Alternatively, the reweighted L_1 regularization approach proposed in [13] encourages both zero weights and weights with small absolute values. In particular, this regularization technique penalizes smaller absolute value weights so that they are driven towards zero faster, resulting in more weights at or near zero. We also incorporate a pruning method [31] to prune weights with absolute values below a certain threshold. Weights near this threshold that remain tend to be small so that they are more likely to be eliminated from the logic operations. In particular, reweighted L_1 minimization can be achieved by employing a log-sum penalty term $\log(\|W\|_1 + \varepsilon)$ in both layers. Therefore, the reweighted L_1 regularization loss $\mathscr{L}_R$ is formulated as follows:

$$\begin{aligned}\mathscr{L}_R &= \lambda_1 \mathscr{L}_{R_1} + \lambda_2 \mathscr{L}_{R_2} \\ &= \lambda_1 \sum_{j=1}^{m} \sum_{i=1}^{D} \log(|w_{1,i,j}| + \beta) + \lambda_2 \sum_{j=1}^{m} \log(|w_{2,j}| + \beta),\end{aligned} \tag{4.14}$$

where $w_{1,i,j}$ and $w_{2,j}$ are the weights of the Rules layer neurons and the Label Layer output neuron, respectively, and $\beta > 0$ is a small value added to ensure numerical stability (e.g., $\beta = 0.01$).

4.6 Experimental Evaluation

In this section, we evaluate DR-Net and NN-Net with both sparsity-promoting mechanisms proposed in Section 4.5: DR-Net with L_0 regularization (DR-Net-L0), DR-Net with reweighted L_1 regularization (DR-Net-RE), NN-Net with L_0 regularization (NN-Net-L0), and NN-Net with reweighted L_1 regularization (NN-Net-RE). Specifically, for all four variations, we show the convergence of networks in the training process, analyze the effectiveness of the sparsity-promoting mechanisms in reducing the rule set complexity, demonstrate the high predictive performances of the proposed DRNet and NNNet, and compare with other state-of-the-art rule learning methods in terms of the accuracy-complexity trade-off.

The numerical experiments were evaluated on four publicly available tabular learning datasets, all of which contain more than 10,000 training instances. The first two datasets are from the UCI Machine Learning Repository [27]: Adult Census (adult) and MAGIC Gamma Telescope (magic). These datasets have also been used in recent works on decision rule learning [24, 91, 25]. The remaining two datasets are the FICO HELOC dataset (heloc) [14] and the home price prediction dataset (house) [89]. As with prior works [91, 25] compared in our evaluation, categorical and numerical attributes are first binarized using well-known encodings, as explained in Section 4.2.1.

The results of the experiments typically include test accuracies and complexities of the derived decision rule sets. Three types of complexities of the decision rule sets are considered in the experiments: number of rules, rule complexity, and model complexity, which capture different aspects of the decision rule models. We define model complexity to be the number of rules plus the total number of conditions in the rule set and rule complexity to be the average number of conditions in each rule of the model.

Unless specified otherwise, both DR-Net and NN-Net were trained for 2000 epochs using the Adam optimizer with a fixed learning rate of 10^{-2} and no weight decay across all experiments. For simplicity, the batch size was fixed at 400, the weights of Rules Layer neurons were uniformly initialized within the range between 0 and 1, and the weights of the Label Layer output neuron were initialized to be all 1. For L_0 regularization, we used the same parameters as proposed in [58]. For reweighted L_1 regularization, the pruning threshold was set to be one tenth of the layer's standard deviation and β was selected to be 0.01.

4.6.1 Training Analysis

We first analyze different behaviors of all four methods (DR-Net-L0, DR-Net-RE, NN-Net-L0, and NN-Net-RE) in the training process and then explored the possible influences that different hyper-parameters can have over the derived decision rules. The results in this subsection were obtained by training all four methods on the adult dataset and each network was initialized with 1000 neurons in the Rules layer to allow sufficient modeling capability.

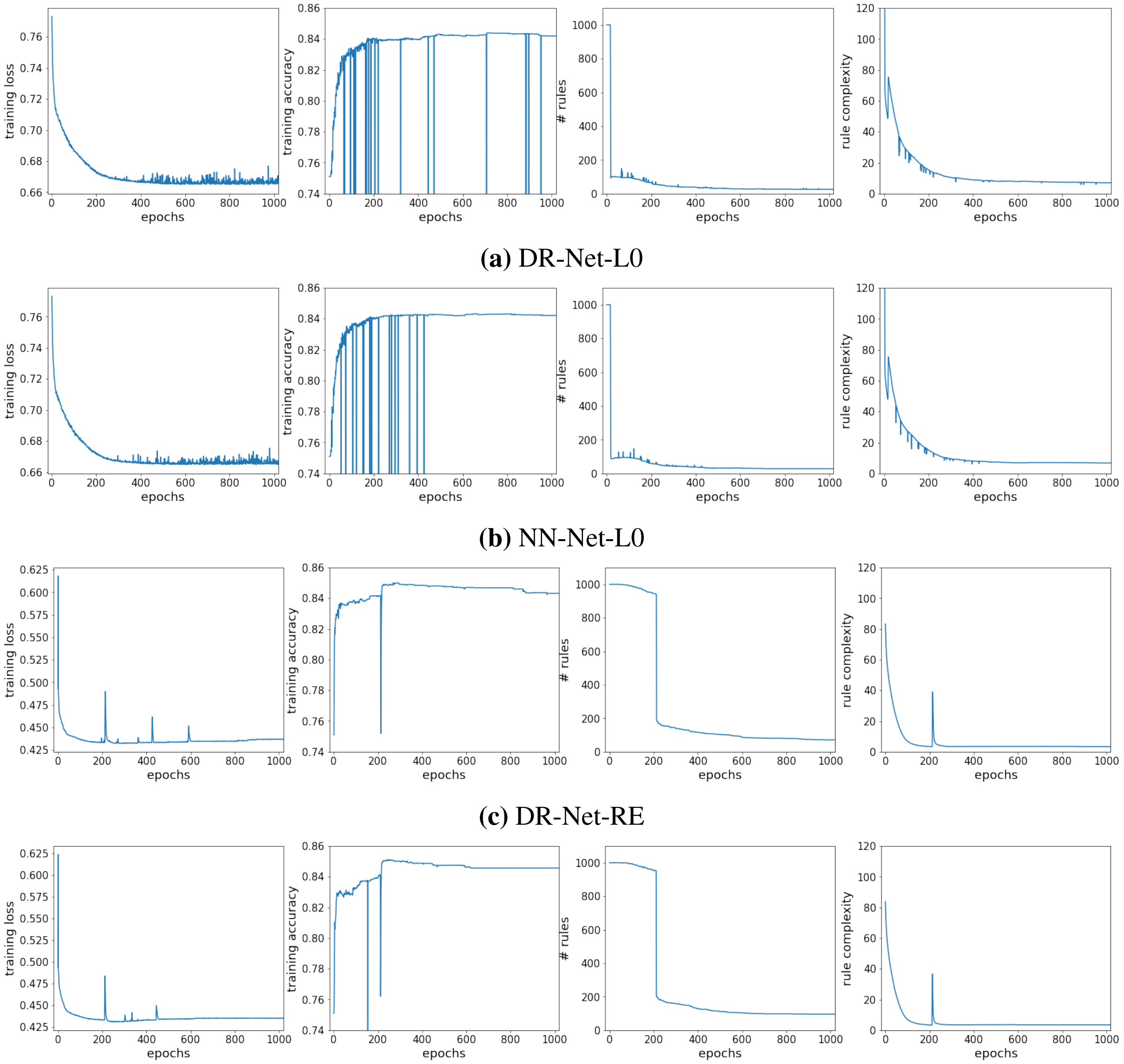
Training Convergence and Complexity Reduction. In this experiment, we empirically demonstrate that the ideas of dynamic bias and the binary step activation function with the modified straight-through estimator can work together to ensure a smooth training procedure. Figure 4.3 shows four statistics (training loss, training accuracy, number of rules, and rule complexity) recorded in the training processes of all four methods as functions of the training epochs, which were all trained with $\lambda_1 = 10^{-1}$ and $\lambda_2 = 10^{-5}$ on the entire adult dataset. Note that only the first half of the training procedure (first 1000 epochs) is shown in the figure while the second half was omitted from the figures to save space, as we noticed that the four statistics for all networks were very stable after 1000 epochs.

First, we notice that DR-Net (DR-Net-L0 and DR-Net-RE) and NN-Net (NN-Net-L0 and NN-Net-RE) have very similar curves for all metrics, which experimentally prove that their architectures are interchangeable and that our neuron designs truly mimic the operations of the Boolean logic gates. The overall trends for all plots match our expectations. In particular, the training losses, the number of rules, and the rule complexities decrease with the increasing number of epochs until convergences at around epoch 400 for all methods. Furthermore, the training accuracies increase from 0.75 (all predictions are 0) to around 0.84 as the number of epochs increases.

However, there are many hard dips in the training process of the networks with L_0 regularization where the training accuracies suddenly drop to around 0.25 and resume back in the next epoch. Some minor spikes and dips are also noted in the plots of the number of rules and the plots of the rule complexity respectively, which exactly correspond to the hard dips found in the plots of training accuracy. These spikes and dips happened in the training process because a weight in the Label Layer was trained to be negative and thus the corresponding rule in the Rules Layer was automatically converted to a set of rules with a single negated feature in each rule according to De Morgan's Laws as discussed in Section 4.3. The conversion from the negation of a rule to a set of rules with a single feature in each rule explains the spikes in the number of rules and dips in the rule complexities. Since every instance has a very high probability to be covered by at least one of the converted rules that have the single negated feature, the networks will predict all instances to be the positive class, which results in the hard dips in the training accuracies. However, as we can see from the plots, continuing training with only one more epoch can conveniently fix the error and thus normally there won't be any negative weights in the Label Layer in practice if all Label Layer weights were initialized with positive weights.

Lastly, we note that the number of rules decreased dramatically in the first few epochs of the training for DR-Net-L0 and NN-Net-L0, as most of the rules were obviously redundant and pruned simultaneously. The behavior of a significant decrease in the number of rules can also be observed for the networks trained using reweighted L_1 regularization at around epoch 200, which explains the disturbances found in the rest of the plots in Figure 4.3 (c) and Figure 4.3 (d).

Effects of sparsity-promoting mechanisms. As discussed in Section 4.5, the rule set complexities can be adjusted by setting different combinations of regularization coefficients λ_1 and λ_2. In theory, the regularization coefficient for the Rules Layer λ_1 should affect the number of conditions per rule, which is captured by rule complexity, and the regularization coefficient for the Label Layer λ_2 should influence the total number of rules. We explore in this experiment how effective the two proposed sparsity-promoting mechanisms are by changing the regularization coefficients and see how the rule complexity and number of rules change, respectively. In particular, we trained all four methods with 5 different λ_1 and 5 different λ_2 for each method on



(a) DR-Net-L0

(b) NN-Net-L0

(c) DR-Net-RE

(d) NN-Net-RE

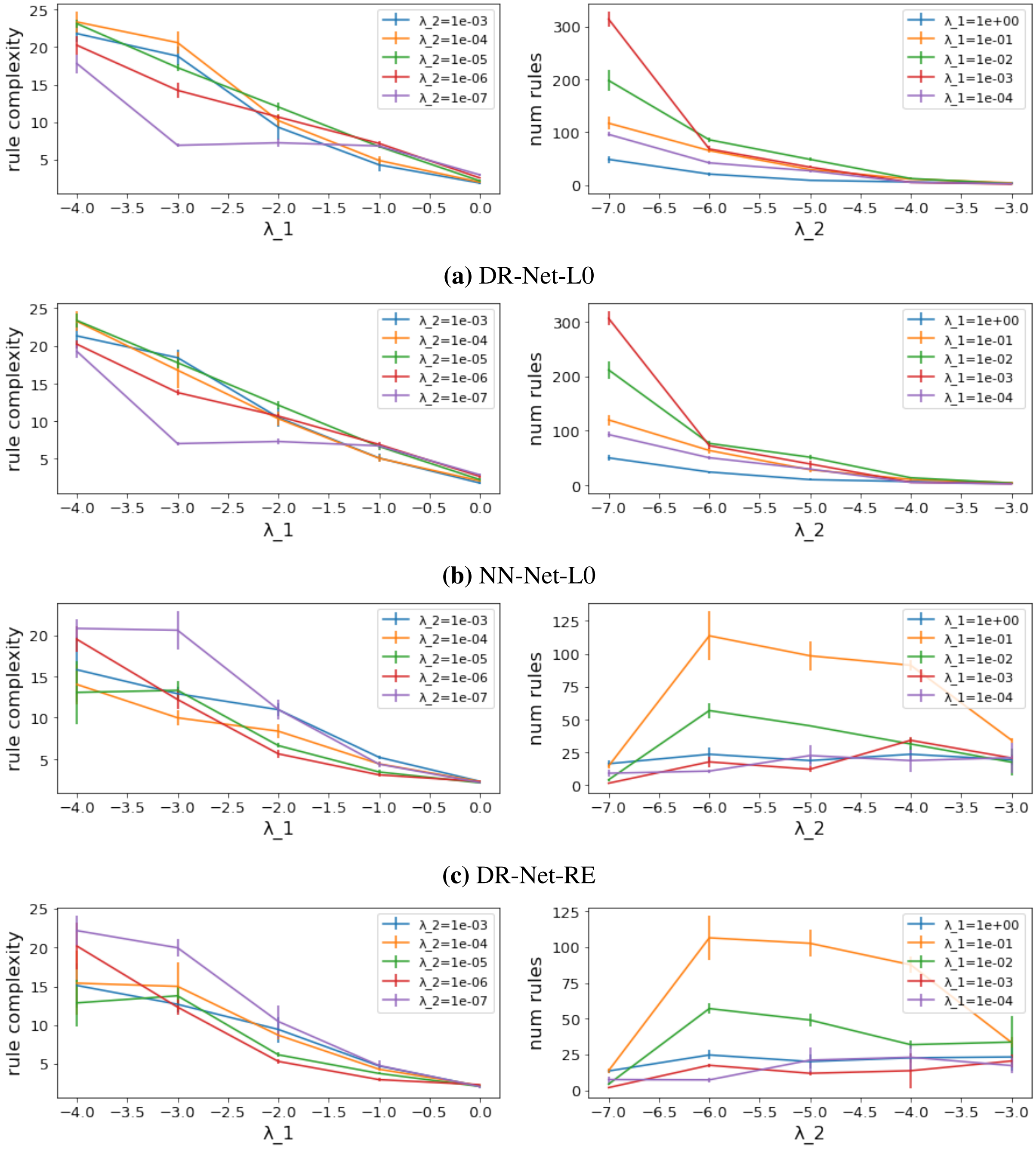
Figure 4.3. Training statistics (training loss, training accuracy, number of rules, and rule complexity) as functions of the number of epochs in the training process.

the adult dataset using 5-fold cross validation. The average complexity values (rule complexity and number of rules) and their standard deviations for different regularization parameters (λ_1 and λ_2) are shown in Figure 4.4.

In the left column of Figure 4.4, each line in the plots shows the changes of the rule complexity as λ_1 increases for a fixed value of λ_2. For all four methods, the rule complexities monotonically decrease as λ_1 increases from 10^{-4} to 1 (note that the values in x-axes are all in log scales), showing the high effectiveness of the regularization term for the Rules Layer among all four methods. The good correlation between the rule complexity and λ_1 for all methods provides our methods a foundation for the excellent accuracy-complexity trade-off capabilities. Similarly, the lines in the right column of Figure 4.4 indicate how the number of rules of the derived decision rule set is changed with respect to λ_2 given that the λ_1 is fixed for each line. The number of rules also decreases monotonically as λ_2 increases from 10^{-7} to 10^{-3} for the networks with the L_0 regularization, while there is no clear relation between the number of rules and λ_2 for networks trained with reweighted L_1 regularization. Thus, compared to networks with reweighted L_1 regularization, networks trained using L_0 regularization demonstrate better potentials of reducing number of rules, as larger λ_2 will consistently result in fewer rules. However, training networks with reweighted L_1 regularization should still be considered as a valid approach, since even the maximum value of the number of rules for DR-Net-RE and NN-Net-RE shown in Figure 4.4 has been pruned to be less than 125, which is a substantial reduction compared to the starting point of 1000 rules.

4.6.2 Classification performance and Interpretability

Next, we demonstrate the advantages of our proposed methods in terms of both predictive performance and rule set interpretability by comparing them with three other state-of-the-art rule learning algorithms: the RIPPER algorithm (RIPPER) [20], Bayesian Rule Sets (BRS) [91], and the Column Generation algorithm (CG) [25]. On the other hand, BRS and CG are examples of recent works in rule learning literature that explicitly consider the interpretability in the training



(a) DR-Net-L0

(b) NN-Net-L0

(c) DR-Net-RE

(d) NN-Net-RE

Figure 4.4. The relations between complexities (rule complexity and number of rules) and regularization parameters (λ_1 and λ_2). All x-axes are in log10 scales. All complexity values were averaged over 5 cross-validation partitions and the vertical bars represent standard deviations.

process. We used open-source implementations on GitHub for all three algorithms, where the CG implementation [3] is slightly modified from the original paper.

In running the experiments of comparing our methods with the decision rule set learning algorithms mentioned above, we only tuned the hyper-parameters that directly affect the interpretability of the rule sets. In particular, we varied the maximum number of conditions and the maximum number of rules, as they are provided by the implementation of RIPPER to constrain the overall complexity of the final model. For BRS, we modified the prior multiplier κ that affects the probability of selecting rules with different lengths, which was also used in [25]. The official implementation [3] of the column generation algorithm (CG) [25] provides two hyper-parameters to set the costs of adding a rule and a condition and thus they instead of the complexity bound parameter C as described in the paper were turned in the experiments. Lastly, for DR-Net-L0, NN-Net-L0, DR-Net-RE, and NN-Net-RE, combinations of λ_1 and λ_2 were varied.

Maximizing Accuracy. To show the upper limits of the predictive performances of our proposed methods, we also included three traditional machine learning methods: Classification and Regression Trees (CART) [9], Random Forests (RF) [10], and a deep neural network (DNN). We used in the experiment scikit-learn [69] implementations for both CART and RF, for which the maximum depth of trees was fixed to be 100 to achieve better generalization. DNN consists of 6 fully connected layers with the ReLU function as the activation function between the layers and each hidden layer of DNN has a fixed number of 50 neurons to ensure enough learning capacity, which was trained with 10000 epochs, a batch size of 2000, a learning rate of 10^{-2}, and a weight decay of 10^{-2}. Note that RF and DNN are typically considered as *uninterpretable* models, which serve as baselines and benchmarks of what black-box models can achieve on the datasets.

To ensure a fair comparison among all rule learners, we leveraged 5-fold nested cross validation to select the best set of hyper-parameters for each rule learner on each dataset that maximized the validation accuracy. Also, the ranges of the hyper-parameters to be varied for each method were purposely constrained so that the learned decision rule sets are fairly interpretable to the users. Thus, for DR-Net-L0, DR-Net-RE, NN-Net-L0, and NN-Net-RE, the number of neurons in the Rules Layer was chosen to be only 50. The test accuracy results of all models on all datasets are shown in Table 4.1. Note that all results in Table 4.1 for CG, BRS, RIPPER, CART, RF, and DNN are copied from [71].

It can be seen in Table 4.1 that across all datasets, all variants of DR-Net and NN-Net outperform other interpretable models in terms of the testing accuracy in most cases with few exceptions. The method with the overall best testing accuracy (NN-Net-L0) can achieve a very similar predictive performance compared to uninterpretable models (RF and DNN) with only a 3% discrepancy. As expected, there is no noticeable difference between the variants of DR-Net and NN-Net in terms of the accuracies, since their architectures are essentially established based on the same logic operations. The CART decision tree algorithm turns out to be the worst performing model in our experiments, which might result from overfitting. The results in Table 4.1 suggest that our proposed methods are very competitive as a machine learning model for interpretable classification.

Accuracy-complexity trade-off. Finally the accuracy-complexity trade-offs abilities were evaluated among all decision rule learning methods that include DR-Net-L0, DR-Net-RE, NN-Net-L0, NN-Net-RE, RIPPER, BRS, and CG. Different sets of accuracy-complexity pairs were generated for each method on each dataset by running the algorithm with a wide range of hyper-parameter values. We ran the experiments on all datasets and the results with the average of the 5-fold cross validation are shown in Figure 4.5 and Figure 4.6. For each method compared, the dots connected by the line segments shown correspond to Pareto efficient models where all other points below the Pareto frontier have either lower accuracies or higher complexities. Again, all results in Figure 4.5 and Figure 4.6 for CG, BRS, and RIPPER are from [71].

The characteristic of being able to attain a high test accuracy with an acceptable model complexity for DR-Net and NN-Net in Table 4.1 is carried over to Figure 4.5 and Figure 4.6.

Table 4.1. Test accuracy based on the nested 5-fold cross validation (%, standard error in parentheses).

dataset	magic	adult	heloc	house
interpretable				
NN-Net-RE	84.14	82.59	70.76	84.89
	(0.69)	(0.23)	(0.66)	(0.46)
NN-Net-L0	84.45	83.13	70.71	86.17
	(0.51)	(0.59)	(0.67)	(0.50)
DR-Net-RE	84.63	82.50	71.05	84.84
	(0.53)	(0.68)	(0.57)	(0.66)
DR-Net-L0	84.10	83.09	70.07	85.90
	(0.82)	(0.51)	(0.89)	(0.50)
DR-Net	84.42	82.97	69.71	85.71
	(0.53)	(0.51)	(1.05)	(0.40)
CG	83.68	82.67	68.65	83.90
	(0.87)	(0.48)	(3.48)	(0.18)
BRS	81.44	79.35	69.42	83.04
	(0.61)	(1.78)	(3.72)	(0.11)
RIPPER	82.22	81.67	69.67	82.47
	(0.51)	(1.05)	(2.09)	(1.84)
CART	80.56	78.87	60.61	82.37
	(0.86)	(0.12)	(2.83)	(0.29)
uninterpretable				
RF	86.47	82.64	70.30	88.70
	(0.54)	(0.49)	(3.70)	(0.28)
DNN	87.07	84.33	70.64	88.84
	(0.71)	(0.42)	(3.37)	(0.26)

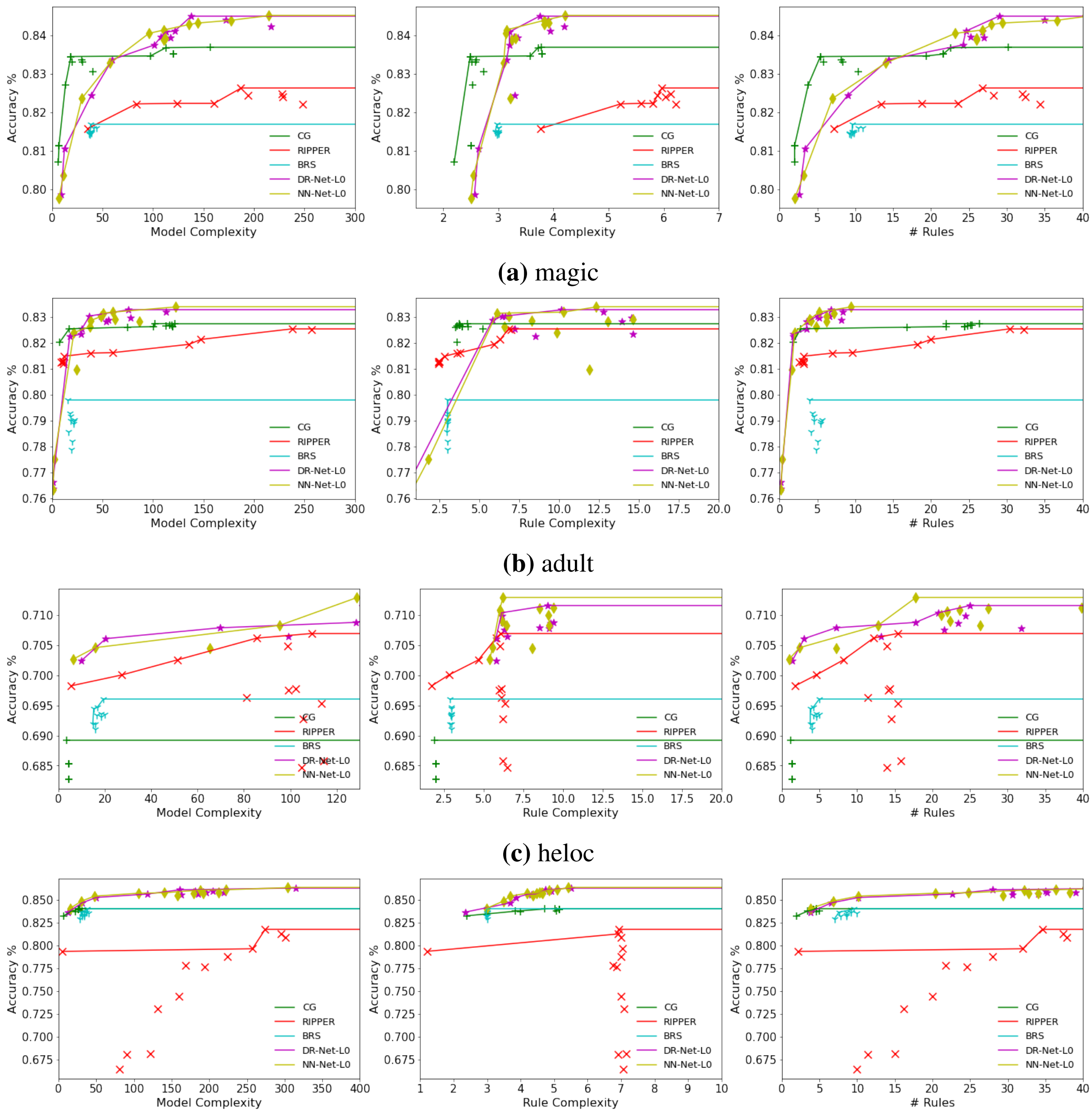
For both the heloc and house datasets, all variants of DR-Net and NN-Net dominate other rule learners on all 3 complexity metrics, as their Pareto efficient points are always on top. For the magic and adult datasets, all variants of DR-Net and NN-Net can still outperform other rule learners in terms of the accuracy by a substantial margin when the model complexity, the rule complexity or the number of rules exceeds a certain threshold. BRS does not demonstrate a clear accuracy-complexity trade-off as its results all group in a very narrow range, which is also noted and explained in [25]. This experiment shows that DR-Net and NN-Net can be preferred over other rule learners because of its potential for achieving a much higher test accuracy with a relatively moderate complexity sacrifice.

4.7 Related Work

Decision rule learning has been extensively studied in the literature, most of which employs heuristic algorithms, but earlier methods optimize for criteria that are not necessarily directly related to classification accuracy or model simplicity. Examples include association rule mining and classification [19, 54], logical analysis methods [23, 8], and greedy set covering approaches [20].

Recently, researchers have improved on decision rule learning algorithms by explicitly considering the interpretability of rules in designing algorithms. In particular, one solution of incorporating interpretability is to add model complexity to the optimization objective so that simplicity can be jointly optimized together with prediction accuracy. Some methods in this category select rules from a set of candidate rules and thus a rule-mining algorithm is employed in the preprocessing step for these methods. Examples include a Bayesian framework that is approximately solved using simulated annealing [91] and an optimization problem solved by a local search algorithm [47]. However, the requirement of starting with pre-mined rules limits the overall search space and their ability to find a globally optimized solution.

There are other methods based on Integer-programming (IP) formulations that do not



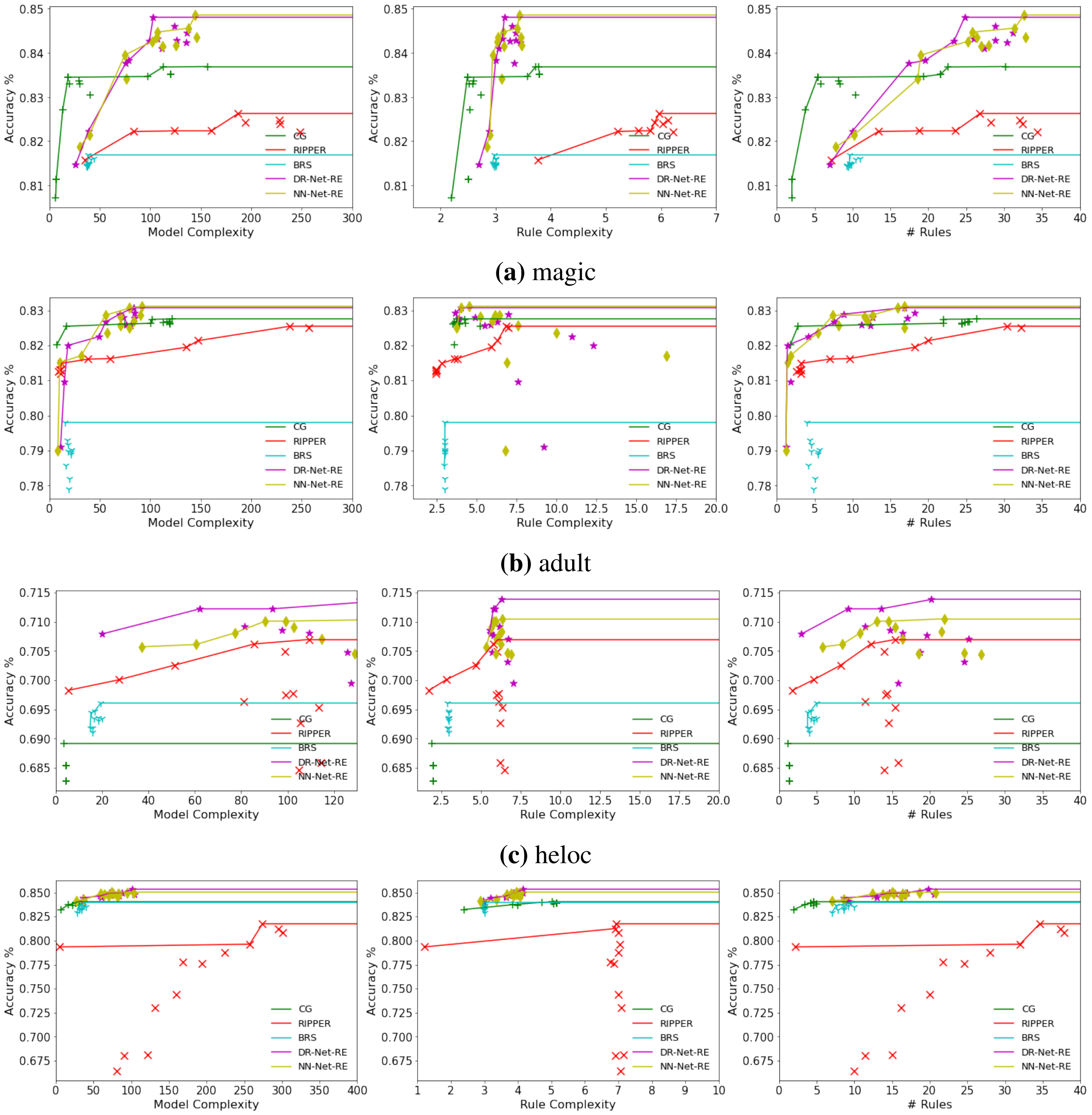
(a) magic

(b) adult

(c) heloc

(d) house

Figure 4.5. Accuracy-Complexity trade-offs on all datasets for DR-Net and NN-Net trained using L_0 regularization. Pareto efficient points are connected by line segments.



(a) magic

(b) adult

(c) heloc

(d) house

Figure 4.6. Accuracy-Complexity trade-offs on all datasets for DR-Net and NN-Net trained using reweighted L_1 regularization. Pareto efficient points are connected by line segments.

require the pre-mining of the rules, but only approximate solutions can be found for large datasets due to the inherent complex nature of the problems. For example, in [25], the IP problem is approximately solved by relaxing it into a linear programming problem and applying the column generation algorithm. In [86], various optimization approaches are utilized, including block coordinate descent and alternating minimization algorithm.

Besides decision rule sets, decision lists [74, 6, 48] and decision trees [9, 75] are also explainable rule-based models. Decision lists capture rules in an ordered IF-THEN-ELSE sequence. However, the cascading of rules in an IF-THEN-ELSE sequence means that the interpretation of an activated rule will unfortunately require an understanding of all preceding rules, which makes the explanation more difficult for humans to understand. Decision trees implicitly organize rules into a tree structure, corresponding to paths in the tree. However, these rules are typically more complex, and thus decision trees are often prone to overfitting.

Building on the notable success that deep neural networks have had on perceptual learning tasks like image classification, researchers have also recently turned to neural network models for tabular data learning [43, 2]. The works in [43, 2] aim to capture aspects of gradient boosting decision trees and random forests that have made these models successful, and they are able to achieve comparable performance as these approaches with neural models. However, like gradient boosting decision trees and random decision forests, these models are also uninterpretable in the sense that they do not provide explanations that are easily understandable by humans.

4.8 Conclusion

In this paper, we extended recent work on decision rule learning based on neural net architectures that can be accurately trained for tabular data classification. In particular, we presented alternative formulations to trainable Boolean logic operators as neurons with continuous weights, including trainable NAND neurons. These alternative formulations provide uniform treatments to different trainable logic neurons so that they can be trained the same way. This enables for example the direct application of existing sparsity-promoting neural net training techniques like reweighted L_1 regularization to derive sparse networks that translate to simpler decision rule sets, in addition to a stochastic L_0 regularization approach that was previously used in [71]. Further, we presented an alternative network architecture based on trainable NAND neurons by applying De Morgan's Law to realize a NAND-NAND network instead of an AND-OR network. Our experimental results show that these alternative formulations can also generate accurate decision rule sets that achieve state-of-the-art performance in terms of accuracy in tabular learning applications.

Furthermore, since all our proposed AND, OR, and NAND neurons are now uniformly defined, layers of different neurons can be freely concatenated using a different order than the ones proposed in this paper. For example, a neural network that directly translates to a Conjunctive Normal Form classifier can be easily formulated by concatenating a layer of OR neurons with an output AND neuron. It would be an interesting direction of future research to see if different layers of logical neurons can be combined in more complicated ways to achieve better predictive performances while maintaining good interpretability.

4.9 Acknowledgements

Chapter 4, in full, is a reprint of the material as it appears in Mach. Learn. Knowl. Extr., 2023. Litao Qiao, Weijia Wang, and Bill Lin. The dissertation author was a co-author of this paper.

Chapter 5

Trained Biased Number Representation for ReRAM-Based Neural Network Accelerators

5.1 Introduction

Convolutional neural networks (CNNs) have achieved breakthrough performance on a variety of artificial intelligence applications, including image classification, video object tracking, natural language processing, two-player games, and autonomous-driving vehicles. However, to continue breakthrough performance on increasingly complex artificial intelligence problems, CNNs have steadily increased in complexity, with recent CNNs requiring over 16 billion floating point operations for a single inference across a deep network with nearly 140 million parameters [32, 84].

Although conventional processor architectures provide plenty of processing power for training deep CNNs, they are often not well-suited for deployment in mobile and wearable applications where energy-efficiency is paramount. In particular, conventional processor architectures typically require frequent data movements between the processor and off-chip memory, which consume enormous amounts of energy. Moreover, although not as significant as the energy cost for data movements, the tens of billions of full-precision floating point operations per inference are also often cost prohibitive in terms of energy consumption.

Recently, there has been considerable excitement surrounding the use of emerging non-volatile memory technologies for the implementation of neural network accelerators. In particular, recent efforts have demonstrated that metal-oxide resistive random access memory (ReRAM) [92] can be used to efficiently implement crossbar structures that provide both storage and computation capabilities. For neural network computations, ReRAM crossbars can be used to both store synaptic weights as well as perform matrix-vector multiplications directly in the analog domain [34, 33, 49, 70, 45, 16, 11, 55, 87, 93]. A number of promising dataflow-like ReRAM-based neural network accelerator architectures (e.g., ISAAC [80], PRIME [17], and PipeLayer [85]) have been proposed that show a substantial advantage in energy-efficiency over conventional processor architectures.

Although an ReRAM crossbar can directly perform matrix-vector multiplication, several critical challenges are presented to ReRAM-based neural network acceleration:

- The precision of weights that can be stored in the crossbar is limited by the resolution of the ReRAM cells, and the precision of inputs to the crossbar is limited by the resolutions of the Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) that are used at the crossbar interface. In particular, practical implementations of ReRAM crossbars are limited to some *m*-bit weight precision and some *p*-bit input precision. For example, in ISAAC [80], the weight precision is $m = 2$ bits, and the input precision is just $p = 1$ bit. In PRIME [17], the weight precision is $m = 4$ bits, and the input precision is $p = 3$ bits. In PipeLayer [85], a spike-based scheme is used in which the inputs are provided as spikes, which eliminates the need for DACs. This effectively corresponds to an input precision of just $p = 1$ bit, while ADCs are replaced with integrate and fire units. PipeLayer supports a weight precision of $m = 4$ bits. Higher precision inputs can be achieved by evaluating the ReRAM crossbar multiple times with successive *p*-bit inputs. For example, both ISAAC [80] and PipeLayer [85] support 16-bit inputs by evaluating the ReRAM crossbar one bit at a time successively sixteen times. However, this way of achieving higher precision inputs increases the processing time. To increase the precision of weights, a group of multiple crossbars can be used, where the j^{th} column of each crossbar in the group logically implements a portion of the weights for the same kernel. However, both means of increasing input and weight precisions cost proportional increases in energy consumption. Moreover, the use of multiple crossbars to increase weight precision limits the size of CNNs that can be implemented as weights are persistent in ReRAM-based neural network implementations.

- Ideally, we would like to use native ReRAM cell precisions of $m = 2$ to 4 bits. However, as observed in [85], the accuracies of ReRAM-based neural network accelerators are sensitive to weight precisions. For deep CNNs on complex datasets, accuracies drop sharply when weight and activation precisions are decreased to low bit-widths. In particular, the full-precision VGG-19 network achieves about 6.7% test error on CIFAR-10 dataset, while this error dramatically increases to 90% if we simply "truncate" it into 4-bit weights and activations. Alternatively, CNNs can be trained with low precision weights and activations [61, 35, 96, 22, 30, 42, 12, 90, 98]. In particular, in [17], a low precision number representation called dynamic fixed point [22, 42] is used for ReRAM-based accelerators, in which an m-bit number is viewed as a 2's complement number that is scaled by a power-of-2 fractional scaling factor M: $\{-M \cdot 2^{m-1}, \ldots, 0, \ldots, M \cdot (2^{m-1} - 1)\}$. This means the representation is symmetric in the range of positive and negative numbers. However, when examining actual weights that these number representations are suppose to approximate, we find that the set of weights on a given CNN layer is often not symmetric. Consider the VGG-11 CNN architecture [84] trained on the CIFAR-10 dataset. The distribution of weights on the Conv5-2 layer, which contains about 2.36 million weights, spans the range $[-0.075, 0.128]$, with the positive range 71% larger than the negative range. This leads to poor approximations when using low precision numbers in dynamic fixed point to represent the weights.

- ReRAM crossbar cells can only represent “positive” conductance values. However, neural networks generally require both positive and negative weights. To implement both positive and negative weights in ReRAM-based neural network accelerators, previous approaches [17, 85] have implemented kernels with positive and negative weights as two separate crossbar arrays. This “sign-splitting” approach significantly increases the hardware cost.

In this paper, we propose to use a *trained biased number representation* to approximate low precision weights. In particular, we view an m-bit number as an unsigned integer that is scaled by a fractional scaling factor M and offset by a *biasing* term K. Each m-bit integer therefore represents a number from the set $\{0-K, M-K, \ldots, M \cdot (2^m-1)-K\}$, where the range of positive and negative numbers can be arbitrarily shifted by the biasing term K, and the step size M can be any fractional scaling factor. The parameters M and K can be independently trained on a *per-layer* basis to best approximate the distribution of weights on a given layer.

To illustrate the benefits of our proposed trained biased number representation, let us consider again the weights on the Conv5-2 layer of VGG-11 that have been trained on the CIFAR-10 dataset. For a precision of $m = 2$ bits, we computed the optimal parameters for these weights for dynamic fixed point and our proposed trained biased number representation, and we computed the mean square error of each representation relative to full-precision weights. Figure 5.1 shows the normalized mean square errors, which shows that dynamic fixed point has 49.3% higher mean square errors vs. our proposed trained biased number representation.

The main contributions of this paper are as follows:

- We propose a new low precision quantization approach for CNNs based on a novel trained biased number representation of weights. Our representation can represent both positive and negative numbers for ReRAM-based implementations without the need for separate crossbar arrays. Moreover, the trained biasing term in our approach enables our representation to approximate well sets of weights that have asymmetric ranges of positive and negative numbers.

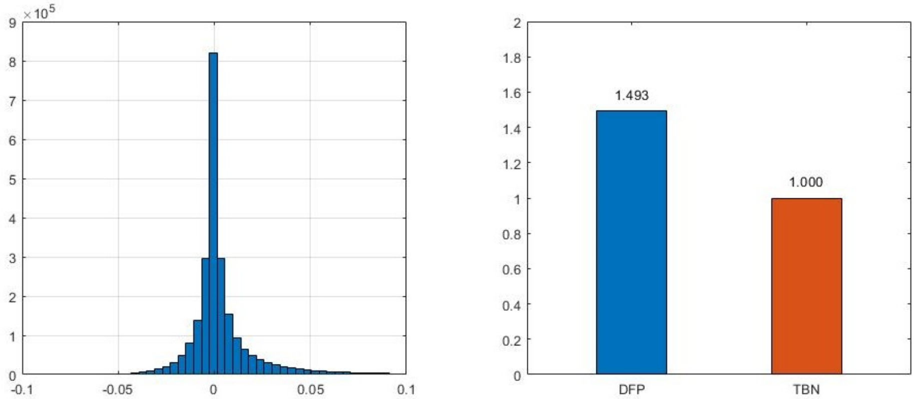


Figure 5.1. The distribution of full-precision weights on the Conv5-2 layer (left) and the comparison of the normalized mean square error between our trained biased number representation (TBN) and dynamic fixed point (DFP) representation for $m = 2$ bits (right).

- Our number representation is well-suited to the inherent matrix-vector multiplication capabilities of ReRAM-based crossbar structures. In particular, our low precision quantization approach can match the resolution limitations of digital/analog converters and memory cells in ReRAM-based crossbars.

- To take full advantage of ReRAM-based analog computational capabilities, the amount of computations that must be performed in the digital domain should be minimized. To this end, we propose a novel activation-side coalescing approach that coalesces the steps of batch normalization, non-linear activation, and quantization into a single stage that simply performs a clipped-rounding operation.

- We explore the configuration space of different combinations of weight precisions and activation precisions by training different versions of the popular VGG deep convolutional neural network architecture [84] on the CIFAR datasets. Experimental results show that our approach substantially outperforms previous low precision number representations and can achieve near full-precision model accuracy with as little as 2-bit weights and 2-bit activations.

The remainder of the paper is organized as follows: Section 5.2 introduces some background on CNNs and ReRAM-based acceleration. Section 5.3 describes our training algorithm for training CNNs with trained biased number representations. Section 5.4 describes our activation-side coalescing approach that combines batch normalization, non-linear activation, and quantization into a single efficient stage. Sections 5.5 and 5.6 present our evaluation results. Section 5.7 concludes the paper.

5.2 Background

5.2.1 Convolutional neural networks

A CNN is a class of deep, feed-forward artificial neural networks that has successfully been applied to multi-channel image classification. A typical CNN comprises a pipeline of connected layers, each performing transformations from a set of input feature maps to a new set of output feature maps. The inputs to the first layer correspond to the channels of an input image, and the outputs of the last layer correspond to the probabilities of classes that best describe that image. Each layer of the CNN is associated with a set of parameters, usually called weights, that are typically trained offline with a labeled dataset. The goal of supervised learning of CNNs is to train these parameters so that the CNN can accurately classify new data points.

In a standard CNN structure, the layers are typically convolutional layers, pooling layers, or fully connected layers. Each convolutional (Conv) layer consists of a number of $h \times w \times C_{in}$ kernels, each of which is convolved with an $H_{in} \times W_{in} \times C_{in}$ multi-channel input feature map to produce the corresponding $H_{out} \times W_{out}$ output channel. Together, a three-dimensional $H_{out} \times W_{out} \times C_{out}$ output feature map is produced from C_{out} kernels. The convolution operation for the z^{th} kernel can be expressed as follows:

$$\mathbf{z}_{out}(x,y,z) = \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} \sum_{k=0}^{C_{in}-1} \mathbf{w}_z(i,j,k) \cdot \mathbf{x}_{in}(x+i,y+j,k) \tag{5.1}$$

The elements of a $h \times w \times C_{in}$ kernel are weights to be trained, and a bias term is usually added to $\mathbf{z}_{out}(x,y,z)$, which is also trained.

A pooling layer maps each input feature map to an output feature map, where each output feature is the maximum or average of an $h \times w$ window of input features. A pooling layer reduces the height and width dimensions of the output feature map by a factor of h and w, respectively. Pooling layers are inserted throughout a CNN to gradually reduce the size of the intermediate feature maps.

A fully-connected (FC) layer takes an input vector and performs a dot product with a weight vector, which can be expressed as follows:

$$\mathbf{z}_{out} = \sum_{i=0}^{C_{in}-1} \mathbf{w}(i) \cdot \mathbf{x}_{in}(i) \tag{5.2}$$

A bias term is also usually added to this output, and this bias term together with the weights is also trained.

For Conv and FC layers, the result of Equation 5.1 or 5.2 is usually passed through a batch normalization (BN) layer [37], which solves the problem of internal covariate shift. The BN operation can be expressed as

$$\mathbf{y} = \gamma \left(\frac{\mathbf{z}_{out} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \right) + \beta \tag{5.3}$$

where μ and σ are statistics collected over the training set, γ and β are trained parameters, ε is used to avoid round-off errors.

Finally, the output of batch normalization is usually passed through a non-linear activation function like ReLU or Sigmoid. In this work, we will assume ReLU activation, which performs $\max(0, \mathbf{y})$.

5.2.2 ReRAM-based crossbar structure

Figure 5.2 depicts an $N \times N$ ReRAM crossbar structure where each ReRAM cell can be programmed with one of multiple possible resistance states. The corresponding conductance of

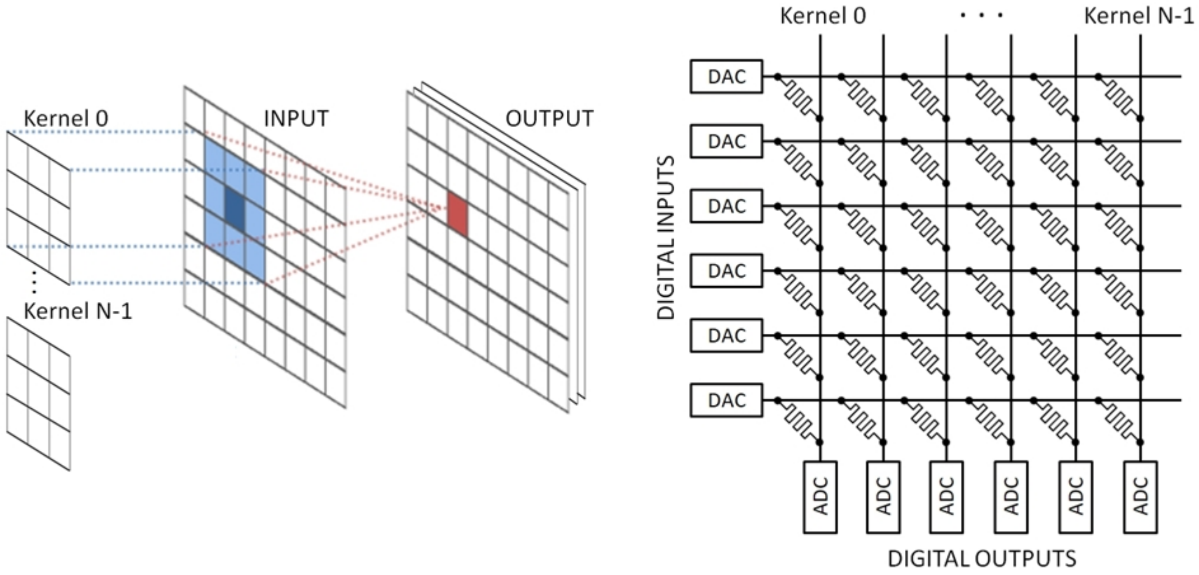


Figure 5.2. The ReRAM-based crossbar structure.

an ReRAM cell at the i^{th} row and j^{th} column of the crossbar is represented by $g_{i,j}$. These ReRAM cells can be used to encode the synaptic weights of a neural net. In the case of a convolutional (Conv) layer in a CNN, each column of the ReRAM crossbar, $j = 0, 1, \ldots, N-1$, can be used to implement a different Conv kernel. The input voltage of each row is represented by v_i, which can be used to encode an input feature of a neural net. Each column j of the ReRAM crossbar can then perform an *analog dot product* of the input voltage vector $\{v_0, v_1, \ldots, v_{N-1}\}$ with the corresponding conductivity vector $\{g_{0,j}, g_{1,j}, \ldots, g_{1,N-1}\}$ as follows:

$$I_{out}^{j} = \sum_{i=0}^{N-1} g_{i,j} \cdot v_i \tag{5.4}$$

These dot product operations across the columns are performed simultaneously as a single matrix-vector multiplication operation. A Digital-to-Analog Converter (DAC) is used to convert a digital input into an analog voltage v_i, and an Analog-to-Digital Converter (ADC) is used to convert an output voltage derived from I_{out}^{j} into a digital output. In the case of a fully connected (FC) layer, the entire crossbar can be used to implement the corresponding weight matrix.

5.3 Learning the biased number representations and weight approximations

5.3.1 Gradient calculations

In our training algorithm, we begin with a pre-trained model with full-precision weights, with each latent full-precision weight denoted as $\tilde{w_i}$. The goal of our training procedure is to assign an m-bit integer $g_i = 0, 1, \ldots, 2^m - 1$ to each $\tilde{w_i}$ so that the latent full-precision weight can be approximated with the following biased number representation:

$$\hat{w_i} = Mg_i - K \tag{5.5}$$

Here, M is the scaling factor, and K is the *biasing* term that gets subtracted from the scaled term Mg_i. Note that in this biased number representation, both positive and *negative* numbers are represented using the same m-bit integers $g_i = 0, 1, \ldots, 2^m - 1$. However, unlike signed fixed point representations that represent a symmetric range of positive and negative numbers, our utilization of a biasing term K allows us to *asymmetrically* partition the range of positive and negative numbers. Further, the scaling factor M allows us to provide the *appropriate resolution* for approximating full-precision weights. For example, for $m = 2$ bits, $M = 0.541$, and $K = 1.182$, $g_i = 0, 1, 2, 3$ correspond to the approximate weights $\hat{w_i} = -1.182, -0.641, -0.1, 0.441$, respectively.

The key idea in our training procedure is that M and K are independent parameters that are trained together with other parameters, including the latent full-precision weights. These independent parameters M and K are defined on a *per-layer* basis, meaning that a different pair of parameters is used for each layer to approximate the latent full-precision weights.

During each feed-forward pass, we assign g_i to $\tilde{w_i}$ as follows:

$$g_i = \text{clip}\left(\text{round}\left(\frac{\tilde{w_i} + K}{M}\right), 0, 2^m - 1\right) \tag{5.6}$$

where

$$\text{clip}(x, x_{\min}, x_{\max}) = \max(x_{\min}, \min(x, x_{\max})) \tag{5.7}$$

During backpropagation, we calculate the gradient for the scaling factor M as follows:

$$\frac{\partial L}{\partial M} = \sum_{i, g_i \neq 0} \frac{1}{g_i} \frac{\partial L}{\partial \hat{w}_i} \tag{5.8}$$

where L is the loss to be optimized.

For the biasing term K, we calculate its gradient as follows:

$$\frac{\partial L}{\partial K} = -\left(\sum_i \frac{\partial L}{\partial \hat{w}_i} \right) \tag{5.9}$$

Finally, the gradient that we use to update each latent full-precision weight is simply the gradient of the corresponding approximate weight:

$$\frac{\partial L}{\partial \tilde{w}_i} = \frac{\partial L}{\partial \hat{w}_i} \tag{5.10}$$

Since the latent full-precision weights $\tilde{w}_i$ are updated together with the independent parameters M and K during backpropagation, a different g_i may get assigned to approximate $\tilde{w}_i$ in the next feed-forward pass. In turn, the new weight approximations $\hat{w}_i$ would be used to derive gradients to update the latent full-precision weights $\tilde{w}_i$ and the independent parameters M and K in the next backpropagation phase. This way, the *biased number representations* are *trained together* with the weights and other parameters of the neural network to minimize classification loss.

The benefits of using a trained biased number representation is that the trained scaling factor provides the appropriate resolution to represent the weights and the trained biasing term provides an asymmetric partitioning of the number range between positive and negative weights. Together, these trained parameters enable our biased number representation to provide more model capacity to the neural network.

5.3.2 The initialization of M and K

As discussed above, M and K are independent parameters that are defined and trained on a per-layer basis. Before we start on the training procedure described in Section 5.3.1, we must first initialize M and K for each layer based on the latent full-precision weights from the pre-trained model. The main idea is that each $g_i = 0, 1, \ldots, 2^m - 1$ defines a separate centroid $\hat{w}_i = Mg_i - K$, and we want to initialize M and K so that the corresponding centroids are linearly spaced across the range of pre-trained full-precision weights in a layer. Let $[r_{\text{min}}, r_{\text{max}}]$ denote this range. Then, we initialize M and K as follows:

$$M = \frac{r_{\text{max}} - r_{\text{min}}}{2^m - 1} \tag{5.11}$$

$$K = -r_{\text{min}} \tag{5.12}$$

Experimentally, we have found that limiting the range of weights to those within two standard deviations from the mean weight of a layer, which covers 95.4% of the weights, leads to a better initialization of M and K. In particular, let $\mu_{\tilde{w}}$ and $\sigma_{\tilde{w}}$ be the mean and standard deviation of the latent full-precision weights in a layer. Then, we define r_{min} and r_{max} as follows:

$$r_{\text{min}} = \mu_{\tilde{w}} - 2\sigma_{\tilde{w}} \tag{5.13}$$

$$r_{\text{max}} = \mu_{\tilde{w}} + 2\sigma_{\tilde{w}} \tag{5.14}$$

Then, M and K are defined accordingly. We have found that, if the number range needs to be increased to minimize loss, then gradient descent can quickly update M and K accordingly to increase the range.

5.4 Activation-Side Coalescing

In the previous section, we described how latent full-precision weights can be accurately approximated using an *m*-bit integer that matches the weight precision of an ReRAM cell. For example, in ISAAC [80], PRIME [17], and PipeLayer [85], the ReRAM cell precisions are 2 bits, 4 bits, and 4 bits, respectively. Besides overcoming the precision challenge of ReRAM cells, we also have to overcome the precision challenge of input precision to the ReRAM crossbar. In ISAAC [80] and PipeLayer [85], a 1-bit input precision is used, whereas a 3-bit input precision is used in PRIME [17]. In general, a p-bit input precision can be used, which means that we are limited to a p-bit activation and intermediate features are store using p-bits. Unfortunately, when p is small, for example $p = 2$ bits, the results for deep CNNs on complex datasets can be prohibitively inaccurate. Higher effective input precision p can be achieved by evaluating the ReRAM crossbar multiple times. For example, an effective input precision of $p = 4$ can be achieved by evaluating the input 2 bits at a time using an ReRAM crossbar with a 2-bit input precision. However, this incurs proportionally more energy and more processing time, both of which are undesirable. Thus, in general, it is important to minimize the number of bits p for representing the activations as long as high accuracy can be maintained. This problem is discussed in this section.

In particular, we first describe in Section 5.4.1 how activations are quantized based on a Gaussian distribution. Then, in Section 5.4.2, we describe an *activation-side coalescing* technique that combines the steps of batch normalization, activation, and quantization into a single stage that simply performs a clipped-rounding operation. We describe in Section 5.4.2 how our trained biased number representation of weights described in Section 5.3 can be combined with activation-side coalescing.

5.4.1 Gaussian-based quantization

To achieve accuracy when using low-bitwidth quantized activations, we use the half-wave Gaussian quantization (HWGQ) approach proposed in [12]. The HWGQ idea is based on the observation that state-of-the-art neural network architectures generally employ batch normalization [37], which forces the responses of each network layer to be a Gaussian distribution with zero mean and unit variance. Moreover, ReLU is widely used as the activation function in state-of-the-art neural network architectures, which acts as an half-wave rectifier that produces linear outputs for non-negative responses. Therefore, the p-bits used to encode the activations only needs to quantized the non-negative range of responses. For example, an activation x_i can be quantized and encoded with an p-bit integer $q_i = 0, 1, \ldots, 2^p - 1$ so that the activation x_i can be approximated with a uniform quantizer as follows:

$$Q(x_i) = \hat{x}_i = Sq_i \tag{5.15}$$

where $\hat{x}_i = 0, S, 2S, \ldots (2^p - 1)S$ are the corresponding quantization levels, which in general can be floating point values since the quantization step S can in general be a floating point number. For a uniform quantizer, we can derive q_i from x_i as follows:

$$q_i = \text{clip}\left(\text{round}\left(\frac{x_i}{S}\right), 0, 2^p - 1\right) \tag{5.16}$$

In [12], an optimal uniform quantizer is derived from a Gaussian distribution with zero mean and unit variance. This is based on the observation that batch normalized network layers generally produce response distributions that are approximately Gaussian with zero mean and unit variance across all units and layers. Therefore, the same uniform quantizer can be used for all activations. There is no need to train a number representation for activations. The activations are indirectly trained by training the batch normalization parameters. In particular, the optimal uniform quantization step S can be derived from a Gaussian distribution with zero mean and unit variance by minimizing the following mean square error

$$\arg\min_{Q} \int \varphi(x)(Q(x)-x)^2 dx \tag{5.17}$$

where $\varphi(x)$ is the corresponding probability density function. The optimal step S can be derived by solving Equation 5.17 by adding the constraint that $Q(x)$ is a uniform quantizer with step S [12].

5.4.2 Combining trained biased weights with activation-side coalescing

In this section, we discuss how our trained biased number representation of weights described in Section 5.3 can be combined with the Gaussian-based quantized activation approach described above in Section 5.4.1. In particular, we wish to store each weight as an m-bit integer g_i and each activation as a p-bit integer q_i, which can be interpreted by the corresponding parameter S for activation and the corresponding per-layer parameters M and K for the weights. For inference, a naïve implementation would operate as follows:

A1. $\hat{x}_i = Sq_i$;
A2. $\hat{w}_i = Mg_i - K$;
A3. $z = \sum_i \hat{w}_i \hat{x}_i + b$;
A4. $y = \text{BatchNorm}(z) = \gamma\left(\frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}\right) + \beta$;
A5. $r = \text{ReLU}(y)$;
A6. $q_{out} = \text{clip}(r/S, 0, 2^p - 1)$;

In Step A6, $2^p - 1$ is the maximum integer for a p-bit integer encoding of the activations.

As can be readily observed in the above inference algorithm, the steps do not match the hardware capabilities of a ReRAM crossbar since the steps involve floating point operations. In particular, a ReRAM crossbar is capable of efficiently computing in the analog domain an integer dot product operation of the form

$$\sum_i g_i q_i \tag{5.18}$$

where each g_i is an m-bit integer that corresponds to the ReRAM cell precision, and each q_i is a p-bit integer that corresponds to the input precision of the ReRAM crossbar. Ideally, we would like to avoid expanding the integers g_i and q_i into floating point numbers in Steps A1 and A2 and performing the dot product in the floating point domain in Step A3. Instead, we would like to perform the dot product in the integer domain to match the hardware capabilities of a ReRAM crossbar. This can be achieved by rewriting Steps A1-A3 as follows:

$$\begin{aligned}
\sum_i \hat{w}_i \hat{x}_i + b &= \sum_i (Mg_i - K)(Sq_i) + b && (5.19)\\
&= S\sum_i (Mg_i q_i - Kq_i) + b && (5.20)\\
&= S\left(M\left(\sum_i g_i q_i\right) - K\left(\sum_i q_i\right)\right) + b && (5.21)\\
&= S(Mp_1 - Kp_2) + b && (5.22)
\end{aligned}$$

where

$$\begin{aligned}
p_1 &= \sum_i g_i q_i && (5.23)\\
p_2 &= \sum_i q_i && (5.24)
\end{aligned}$$

As can be readily observed, p_1 can be directly implemented as an integer dot product using a column in a ReRAM crossbar. In particular, each column in a ReRAM crossbar can be used to implement this p_1 computation for a different kernel, as depicted in Figure 5.2.

The computation for p_2 can also readily be implemented directly as an integer dot product using a column in a ReRAM crossbar by programming the corresponding ReRAM cells with unit weights. However, since the computation of p_2 is *kernel independent*, the p_2 computation can be *shared by all the kernels* that are implemented on the same ReRAM crossbar. For example, if we have a baseline 128×128 ReRAM crossbar, we can add one more column to create a 128×129 array and use the last column to implement p_2, which can be shared by all 128 kernels implemented in the array. Thus, the amortized cost of p_2 is negligible.

Besides performing integer dot products directly using g_i and q_i, we propose to further optimize batch normalization, ReLU activation, and quantization steps (Steps A4-A6) by combining them into a single step. In particular, we wish to eliminate the floating point operations in batch normalization and absorb the batch normalization parameters as well as the quantization step parameter S in Equation 5.22 directly into a simple clipped-rounding operation.

The optimized algorithm is as follows:

B1. $p_1 = \sum_i g_i q_i$;
B2. $p_2 = \sum_i q_i$;
B3. $q_{out} = \text{activation\_side\_coalescing}(p_1, p_2)$;

In particular, given Equation 5.22, we have $z = S(Mp_1 - Kp_2) + b$. Then the batch normalization operation can be stated as follows:

$$\begin{aligned} y &= \gamma\left(\frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}\right)+\beta && (5.25)\\ &= \gamma\left(\frac{SMp_1 - SKp_2 + b - \mu}{\sqrt{\sigma^2+\varepsilon}}\right)+\beta && (5.26)\\ &= \frac{\gamma S(Mp_1 - Kp_2)}{\sqrt{\sigma^2+\varepsilon}} + \frac{\gamma(b-\mu)}{\sqrt{\sigma^2+\varepsilon}} + \beta && (5.27)\end{aligned}$$

Then we can compute

$$\begin{aligned} \hat{q} &= \text{round}(y/S) && (5.28)\\ &= \text{round}\left[\frac{\gamma(Mp_1 - Kp_2)}{\sqrt{\sigma^2+\varepsilon}} + \frac{\gamma(b-\mu)}{S\sqrt{\sigma^2+\varepsilon}} + \frac{\beta}{S}\right] && (5.29)\end{aligned}$$

Given that γ, μ, σ, ε, β, and S are all constants after training, we can pre-compute these constants for all activations:

$$A = \frac{\gamma M}{\sqrt{\sigma^2+\varepsilon}} \tag{5.30}$$

$$B = \frac{\gamma K}{\sqrt{\sigma^2+\varepsilon}} \tag{5.31}$$

$$C = \frac{\gamma(b-\mu)}{S\sqrt{\sigma^2+\varepsilon}}+\frac{\beta}{S} \tag{5.32}$$

Then we have

$$\hat{q} = \text{round}(Ap_1 - Bp_2 + C) \tag{5.33}$$

$$q_{out} = \text{clip}(\hat{q}, 0, 2^p - 1) \tag{5.34}$$

We can directly implement Equations 5.33 and 5.34 in a single clipped-rounding operation in activation_side_coalescing(p_1, p_2). Note that the clip operation effectively performs a clipped ReLU activation. The operations in Equations 5.33 and 5.34 can be easily implemented in the digital domain[1].

5.5 Evaluation

5.5.1 Evaluation setup

We have implemented our proposed training algorithm based on a trained biased number representation described in Section 5.3 and our proposed activation-side coalescing technique described in Section 5.4 in the PyTorch framework [68]. We use the CIFAR-10 and CIFAR-100 datasets to evaluate our solutions on three versions of the popular VGG deep convolutional neural network architecture [84]: VGG-11, VGG-13 and VGG-19. Since VGG networks were originally proposed for the ImageNet dataset whose input size is 224 × 224 × 3, whereas the

[1] Given the clipped-rounding operations to just a few bits, the operands in Equation 5.33 do not need to be full-precision. Experimentally, we found that an 8-bit fixed point representation of the operands is more than enough for small values of p.

CIFAR images are $32 \times 32 \times 3$, we reduce the size of the fully-connected layers to match the input features. The configurations of networks are summarized in Table 5.1. All convolutional and fully-connected layers except the last layer are followed by a batch normalization layer and ReLU non-linearity in sequence. The Adam optimizer is used to update the scaling and shifting factors, with the learning rate initialized to be 1e-6. The other parameters of the network are optimized by stochastic gradient descent (SGD) with a learning rate starting at 0.01. The momentum and weight decay factor that we use for SGD are 0.9 and 1e-4, respectively. We use a mini-batch size of 128 and divide the learning rates for both optimizers by 10 every 75 epochs. The minimal learning rates for SGD and Adam are 1e-7 and 1e-8, respectively. The results are computed by taking the average of the last 7 test accuracy numbers with the maximal and minimal values removed.

5.5.2 Evaluation results

As discussed in Section 5.1, weights can be quantized into low precision representations using dynamic fixed point (DFP) [22, 42] implementations. To evaluate the performance of our trained biased number (TBN) representation, we apply these two methods to compress weights into the resolutions from 2 to 8 bits with features remaining in full precision. The scaling factors of DFP are derived by minimizing the quadratic error from a pre-trained model, while the scaling and shifting factors of TBN are trained according to (5.8) and (5.9) from the same pre-trained model. The results are shown in Table 5.2 together with the full-precision (FP) accuracies added as a baseline.

It can be seen that, compared with DFP, TBN achieves a significantly higher accuracy. On CIFAR-10 dataset, TBN keeps the loss within 1% with 2-bit weights while the performance of DFP drops below 30%, in comparison with full-precision weights. In particular, since there are only 10 classes in total in CIFAR-10, an accuracy of 10% implies that the model is seriously damaged and fails to produce any reasonable classification. When evaluated using CIFAR-100 dataset, 2-bit TBN results in a more significant accuracy drop on VGG-11, which is 2% worse

Table 5.1. CNN configurations. Fully-connected layers have been adjusted to fit the size of CIFAR dataset.

Layer	VGG-11	VGG-13	VGG-19
Conv1	3×3×3, 64	3×3×3, 64	3×3×3, 64
		3×3×64, 64	3×3×64, 64
Max-pool	2×2		
Conv2	3×3×64, 128	3×3×64, 128	3×3×64, 128
		3×3×128, 128	3×3×128, 128
Max-pool	2×2		
Conv3	3×3×128, 256	3×3×128, 256	3×3×128, 256
			3×3×256, 256
	3×3×256, 256	3×3×256, 256	3×3×256, 256
			3×3×256, 256
Max-pool	2×2		
Conv4	3×3×256, 512	3×3×256, 512	3×3×256, 512
			3×3×512, 512
	3×3×512, 512	3×3×512, 512	3×3×512, 512
			3×3×512, 512
Max-pool	2×2		
Conv5	3×3×512, 512	3×3×512, 512	3×3×512, 512
			3×3×512, 512
	3×3×512, 512	3×3×512, 512	3×3×512, 512
			3×3×512, 512
Max-pool	2×2		
FC1	512×512		
FC2	512×512		
FC3	512×10 (100)		

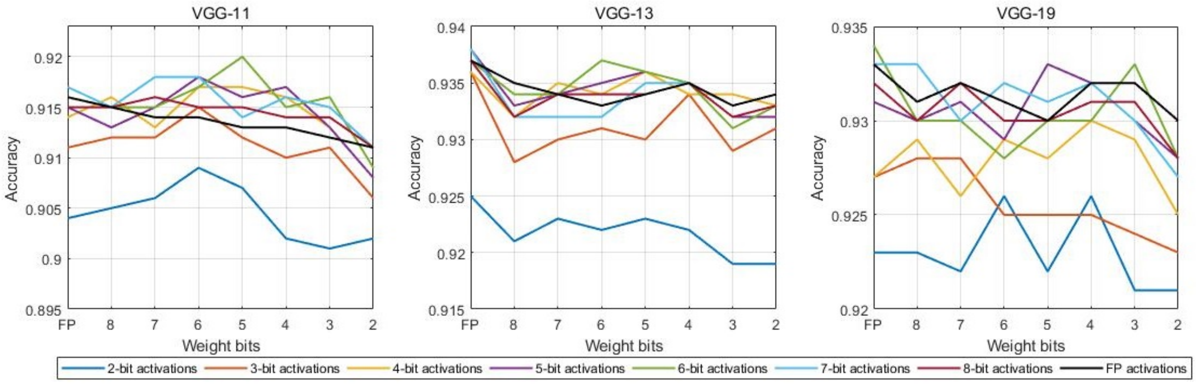


Figure 5.3. Classification performances of VGG-11 (left), VGG-13 (middle) and VGG-19 (right) on CIFAR-10 with different weight and activation precisions.

Table 5.2. Classification accuracy in percentage (%) on CIFAR-10.

Weight bits			2	3	4	5	6	7	8	FP
CIFAR-10	VGG-11	DFP	10.0	71.6	86.9	88.1	90.0	90.0	90.0	91.6
		TBN	91.1	91.2	91.3	91.3	91.4	91.4	91.5	
	VGG-13	DFP	10.0	35.2	87.6	91.6	93.1	93.4	93.3	93.7
		TBN	93.4	93.3	93.5	93.4	93.3	93.4	93.5	
	VGG-19	DFP	10.0	10.0	71.5	90.0	92.7	93.1	93.1	93.3
		TBN	93.0	93.2	93.2	93.0	93.1	93.2	93.1	
CIFAR-100	VGG-11	DFP	1.0	5.0	61.6	67.7	69.0	70.2	70.1	70.2
		TBN	68.2	69.4	69.7	70.1	70.1	70.2	70.2	
	VGG-13	DFP	1.0	4.7	61.3	69.0	72.4	72.7	72.8	73.3
		TBN	72.9	73.2	73.2	73.0	73.1	73.2	73.5	
	VGG-19	DFP	1.0	1.0	36.5	61.7	69.0	70.4	71.2	72.2
		TBN	71.6	72.3	72.2	72.2	72.3	72.3	72.2	

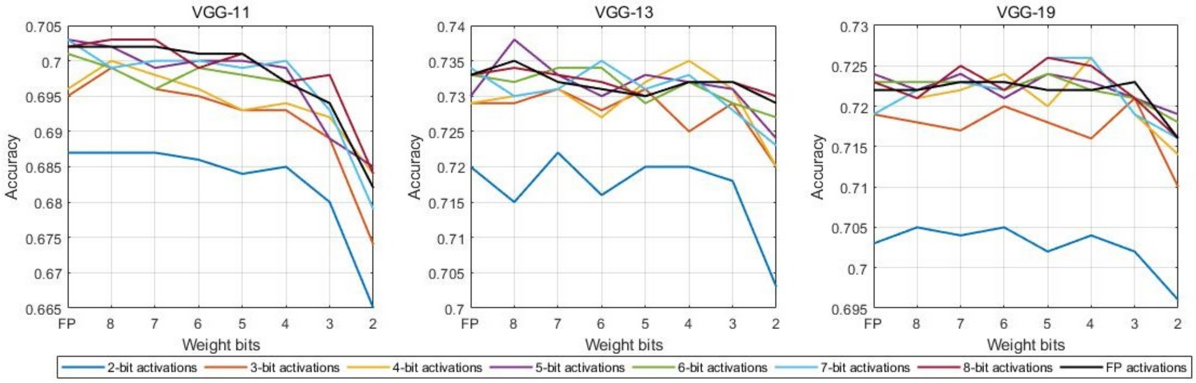


Figure 5.4. Classification performances of VGG-11 (left), VGG-13 (middle) and VGG-19 (right) on CIFAR-100 with different weight and activation precisions.

than the full-precision model, while this loss shrinks sharply with the increase of bitwidth and reduces to 0.5% with 4-bit weights. On the other side, DFP turns to have an extremely poor performance with 2-bit weights and the 4-bit degradation is still larger than 5%, compared with the full-precision models. Moreover, on VGG-11 network and CIFAR-10 dataset, even when using 8-bit representations, DFP still results in an obvious loss in accuracy, due to the fact that the quantization errors from the pre-trained model remain substantial, which determines an upper bound on the accuracy. Here, we note that M and K for TBN are updated according to gradients that are derived to directly minimize classification loss, and that the latent full-precision weights are also trained to compensate for errors caused by low precision.

Next, we compare our trained biased number representation approach with activation-side coalescing on different combinations of precisions for weights and activations on the same VGG network configurations (VGG-11, VGG-13, and VGG-19). In particular, for both weights and activations, we vary the precision from 2 to 8 bits, and we compare each configuration with full-precision accuracy. The results of CIFAR-10 and CIFAR-100 are illustrated in Figure 5.3 and Figure 5.4, respectively, as a function of weight precision with multiple lines to show different activation precisions.

In all cases, the bottom blue line corresponding to 2-bit activations shows the lowest accuracy. However, even with 2-bit activations, our approach achieves accuracies within about 1% of full-precision activations on average under different weight precisions. With 3-bit activations, our approach achieves accuracies within just 0.5% of full-precision activations on average, which is negligible consider the substantial energy savings in ReRAM-based implementations. The slopes of the lines reflect the accuracy loss due to decreasing weight precisions. There exists some ripples in the curves due to the inherent randomness of SGD-based training in the experiments, but it can be identified that, as expected, the accuracy tends to decrease as the precision of weights is reduced. It can be seen that the drops corresponding to CIFAR-100 are sharper than those of CIFAR-10, potentially because of the less redundancy in networks for more complicated datasets. On CIFAR-10, the largest degradation from 2-bit to full precision is only about 0.5%, and when using 4-bit weights and activations, we achieve virtually no loss relative to full-precision models on both CIFAR-10 and CIFAR-100 datasets.

In particular, with 2-bit weights and 2-bit activations, our proposed approach on the CIFAR-10 dataset achieves accuracies of 90.2%, 91.9% and 92.1% on VGG-11, VGG-13 and VGG-19, respectively. Comparing with full-precision model accuracy, which are 91.6%, 93.7% and 93.3%, the accuracy loss is about 1.5%. Moreover, the difference in accuracies vs. full-precision accuracies on both the CIFAR-10 and CIFAR-100 datasets can be further reduced to $< 1\%$ with 3-bit weights and activations and to virtually no loss with 4-bit weights and activations.

5.6 Energy and Area Estimation

In this section, we evaluate the energy and area consumption of our approach based on a recent ReRAM-based hardware accelerator architecture [80]. The analyses are performed on the VGG-11 network trained on the CIFAR datasets[2]. As shown in [80], at 32*nm*, the optimal design point is achieved by using 128×128 ReRAM crossbar arrays with a cell precision of 2 bits, which are considered as the basic unit in our evaluation. To increase the precision of weights, a group of multiple crossbars can be used, which spatially increases both energy and area consumption. Also as proposed in [80], we use an input precision of 1 bit, which effectively replaces the DACs with trivial inverters, and 128 such 1-bit DACs are used for the 128 rows of every crossbar array. Higher precision inputs can be achieved by evaluating the ReRAM crossbar multiple times with successive 1-bit inputs, which temporally increases energy consumption. At the output side of a crossbar array, an 8-bit ADC is shared by all 128 columns. As shown in [80], the cycle time is bounded by the large latency of crossbar arrays, which is on the order of 100*ns*, whereas a frequency on the level of giga-samples-per-second (GSps) can be achieved for an 8-bit ADC. Thus, an 8-bit ADC can be time-multiplexed by the columns of the crossbar

[2] In particular, the energy and area estimations are based on the CIFAR-10 dataset, noting that the difference in overhead between CIFAR-10 and CIFAR-100 is negligibly close to 0 due to the fact that their architectures only differ in the last fully-connected layer.

without performance degradation. Moreover, an additional column per crossbar is also added in [80], and the extra cost has already been accounted for, so there is no need to further re-scale the overhead. For the storage of intermediate features, the architecture proposed in [80] uses on-chip eDRAM buffers. In Table 5.3, we summarize the power and area costs of crossbar arrays and eDRAM buffers, as derived from [80].

We evaluate our trained biased number representation approach (labeled as "TBN") in comparison with a dynamic fixed point approach (labeled as "DFP") using 2-bit weights and activations (namely 2-bit precision for both weights and activations). For the DFP approach, "sign splitting" is required since an ReRAM crossbar cannot directly implement both positive and negative weights when the DFP representation is used, as explained in [17] and Section 5.1. Therefore, two separate crossbars are required to represent positive and negative weights, respectively, and the final results can be obtained by subtracting the outputs of two arrays. The costs of two separate crossbars are reflected in the DFP results. In addition, we include the estimations of a 16-bit fixed point implementation (labeled as "16-bit") to provide a baseline for comparison.

In Table 5.4, we summarize the energy and area results for the three models. In particular, the table reports the number of ReRAM crossbar arrays and the size of the eDRAM buffers for each of the three implementations. As explained in [80, 85], with the data flows fully pipelined, the energy consumption is proportional to the processing time of dot-product operations, whose bottleneck is the layer with the largest latency, which can be up to thousands of cycles. To optimize the critical layers, we adopt the parallelism granularity scheme described in [85]. In particular, for the 16-bit fixed point implementation, the number of ReRAM crossbar arrays reported in Table 5.4 has the Conv1 and Conv2 layers duplicated 16 and 4 times, respectively, to match the speed of other layers. As the input images of the Conv1 layer have a precision of 16 bits, the Conv1 layer for the 2-bit TBN and DFP implementations has to be further duplicated by another factor of $16/2 = 8$ times to match the processing times of the other layers with 2-bit input features. In addition to reporting the absolute energy and area results for the three

Table 5.3. Unit cost of ReRAM array and eDRAM buffer.

ReRAM properties				
Component	Parameter	Spec	Power (mW)	Area (mm^2)
ReRAM array	resolution	2 bits	0.3	0.000025
	size	128 × 128		
ADC	resolution	8 bits	2	0.0012
	number	1		
DAC	resolution	1 bit	0.5	0.000021
	number	128		
Interface[a]			0.319375	0.000787
Total			3.119375	0.002033

eDRAM properties				
eDRAM[b]	size	1 KB	0.432813	0.002703

[a] The interface component comprises the amortized cost of input/output registers and routers to interface with eDRAM buffers and other ReRAM arrays as well as the sample-and-hold and shift-and-add units for data accumulation.
[b] The unit cost of the eDRAM component is provided on a per-page (1 KB) basis, and this unit cost includes the amortized cost of the memory bus for interfacing with ReRAM arrays.

implementations, we also provide in Table 5.4 the normalized results with respect to the 2-bit TBN cost to illustrate our overhead reduction.

As shown in Table 5.4, our TBN approach has lower energy and area costs than both the DFP and 16-bit models. In particular, in comparison to the 16-bit results, 2-bit TBN achieves 6.7× area reduction since it uses fewer crossbar arrays to encode synaptic weights and smaller buffers to store intermediate features. Moreover, the 16-bit model consumes 53.4 times as much

Table 5.4. Area and energy estimations.

Parameter	16-bit	2-bit DFP	2-bit TBN
Number of ReRAM arrays	4948	1352	742
eDRAM size (KB)	298	40	40
Area (mm^2)	10.87	2.86	1.62
Normalized area	6.72	1.77	1
Energy (μJ/img)	1593.72	54.20	29.85
Normalized energy	53.39	1.82	1

energy as TBN does to process an image, due to the quadratic decrease in energy caused by lower power and faster speed of the low precision models. When compared with 2-bit DFP results, TBN achieves 1.8× reduction in both area and energy costs because of the double crossbar arrays introduced in DFP to represent positive and negative weights.

5.7 Conclusions

In this paper, we consider the problem of training convolutional neural networks with low precision weights and activations in a manner that is compatible with an implementation on ReRAM-based neural network accelerators. Low precision weights and activations are needed to match the low resolutions of memory cells and input voltages in ReRAM-based structures. In particular, non-uniform quantization approaches are not amenable to ReRAM-based crossbar implementations, and previous uniform quantization approaches have poor accuracies when applied to deep CNNs on complex datasets. We propose a trained biased number representation with trainable scaling and shifting factors that can approximate well asymmetric number ranges, which can achieve near full-precision model accuracy with as little as 2-bit weights and 2-bit activations on difficult datasets. Moreover, we propose an activation-side coalescing technique that combines the steps of batch normalization, non-linear activation, and quantization into a single stage that simply performs a clipped-rounding operation. Evaluation results show that our trained biased number representation significantly outperforms previous quantization approach in terms of both classification errors and the costs of energy and area. In particular, our models achieve accuracies within about of 1.5% of full-precision model accuracy with 2-bit weights and activations on CIFAR-10 dataset, and about 0.1% of accuracy degradation with 4-bit weights and activations on both CIFAR-10 and CIFAR-100 datasets. Moreover, when using 2-bit weights and activations, our proposed approach yields about 6.7× and 53.4× reduction in terms of area and energy consumption, respectively.

5.8 Acknowledgements

Chapter 5, in full, is a reprint of the material as it appears in ACM Journal on Emerging Technologies in Computing Systems, 2019. Weijia Wang and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Optimizing 3D U-Net based Brain Tumor Segmentation with Integer-Arithmetic Deep Learning Accelerators

6.1 Introduction

In the past few years, brain tumors have become one of the most deadly cancers in the world, especially for relatively young patients. Brain tumors can generally be divided into two types: 1) primary brain tumors that originate in the brain, and 2) secondary brain tumors metastasized from other organs. In particular, gliomas are the most common malignant tumors that account for about 75% of all the brain cancers. Based on the growth potential and aggressiveness of the tumor, gliomas are categorized into four grades – grades I and II are often referred to as "low-grade gliomas (LGG)" while grades III and IV are referred to as "high-grade gliomas (HGG)."

Currently, Magnetic Resonance Imaging (MRI) modalities are the most commonly utilized technique for the brain tumor diagnosis. Various MRI modalities highlight different tissue properties and brain tumors can be further categorized and segmented into multiple subregions. However, radiation therapists manually labeling the scans is burdensome, inefficient, and requires high technical expertise. In this context, there has been an emerging need for automatic brain tumor segmentation and deep learning techniques are introduced due to their recent considerable success in image processing applications.

Multimodal Brain Tumor Segmentation Challenge 2018 (BraTS 2018) is a public benchmark that provides a set of 3D MRI scans with ground truths labeled by human experts and the task is to develop machine learning algorithms to produce the segmentation labels of different glioma sub-regions. In particular, the training dataset of the challenge comprises 210 HGG and 75 LGG MRI cases. Each case consists of four MRI modalities of shape $240 \times 240 \times 155$, including 1) native (T1), 2) post-contrast T1-weighted (T1Gd), 3) T2-weighted (T2), and 4) T2 Fluid Attenuated Inversion Recovery (FLAIR) volumes. The ground truth segmentation is also $240 \times 240 \times 155$ volumetric images, which are manually delineated by one to four raters according to the same annotation protocol. Three labels are provided along with an additional background label, i.e., 1) the necrotic and non-enhancing tumor core (NCR & NET, label 1), 2) the peritumoral edema (ED, label 2), 3) and the GD-enhancing tumor (ET, label 4). Furthermore, the challenge participants are expected to segment the images into three sub-regions: 1) the ET region, 2) the tumor core (TC) that is the combination of NCR, NET and ET, and 3) the whole tumor (WT) that includes all the three tumor labels. In addition, the benchmark provides 66 unlabeled validation cases and 161 unlabeled testing cases, based on which the participating algorithms are evaluated and the final ranking is computed. The number of LGG and HGG subjects are not specified in the validation and testing datasets.

While a number of convolutional neural networks (CNNs) and fully-convolutional networks (FCNs) have been proposed with the growing demand and interest in automatic brain tumor segmentation, a major bottleneck of this application is the volumetric multi-channel modality images that take up significant memory and computational power, which can be expensive even for the latest and most powerful GPUs. For example, a whole multi-modal MRI image of the BraTS 2018 challenge [59] cannot fit into one single GPU and it needs to be cut into patches during training and inference. Furthermore, lighter platforms, e.g., medical devices, generally have more limited on-device memory and computational power. On the other hand, massively parallel deep learning accelerators have been developed to exploit low-bit-width arithmetic. For instance, NVIDIA has provided a NVIDIA Deep Learning Accelerator (NVDLA) framework [64] to address the computational demands of inference, which allows using multiple data types across its various functional units to save area and computational power, including as little as binary integers. Deep neural networks that are available to operate in a low-precision manner can be deployed using deep learning accelerators to improve the chip design by allowing more cores on the chip with limited area. Therefore, the model optimization and acceleration play a critical role for practical deployment.

Basically, storage cost can be reduced by using low-precision parameters whereas cheap computational cost can be achieved by performing low-bit-width arithmetic, which takes advantages from both low-precision weights and activations. However, the performance of a full-precision network can be vulnerable when converting the model into fewer bits. In general, a model gets prohibitively ruined by inferring with directly "truncated" low-bit-width arithmetic. In this work, we propose a quantization technique along with a training strategy that supports the volumetric segmentation with the dot-product operations in an integer-arithmetic manner. The floating-point decoding and encoding phases are deferred until the end of layers.

The rest of this paper is organized as follows: Section 6.2 introduces some neural network architectures for image segmentation and some others' works to compress and accelerate deep neural networks. Section 6.3 describes the formulation and training algorithm of our quantization approach, as well as a procedure to perform integer-arithmetic operations for post-training inference. Section 6.4 evaluates our approach in comparison with the full-precision model. Section 6.5 concludes the paper.

6.2 Related work

6.2.1 Automatic volumetric segmentation

State-of-the-art works employ full-convolutional networks (FCNs) [57] for automatic brain tumor segmentation. Different from other common convolution neural networks that use a fully-connected layer at the end, FCNs also employ a convolutional layer as the last layer to produce a pixel-wise prediction. In particular, a fundamental FCN architecture, namely U-Net [76], consists of a contracting encoder (a.k.a. analysis path) and a successive expanding decoder (a.k.a. synthesis path). The encoding part analyzes the input image and interprets it as a feature map, which is then fed into the decoder. Moreover, high-resolution activations in the analysis path are concatenated with up-sampled outputs in the synthesis path through shortcut connections to achieve better localization performance. Due to the symmetric fully-convolutional architecture, the decoding part constructs a label map with the same size of the input image, each of whose channels corresponds to a segmentation label. Within a channel, every pixel indicates the probability of the corresponding label being positive.

Though U-Nets have achieved an accuracy close to human performance in segmenting 2D images, when it is applied to volumetric medical images, 3D images have to be processed as multiple 2D slices and hence it fails to capture the relationship of adjacent slices. Therefore, some later works further propose volumetric extensions of the U-Net to produce smoother volumetric segmentation.

In particular, the authors of U-Net also propose their feasible solution to the volumetric segmentation problem, namely, 3D U-Net [99], by replacing the 2D convolutions in U-Net their 3D counterparts. An overview of the 3D U-Net is illustrated in Figure 6.1. As can be seen in Figure 6.1, like U-Net, 3D U-Net comprises the left analasis path and the right synthesis path. In particular, each stage of the encoder consists of two 3D convolutional layers with a kernel size of $3 \times 3 \times 3$ and a 3D max pooling layer to down-sample the feature map. On the other side, there are also two $3 \times 3 \times 3$ convolutions at each stage and the up-sampling is performed with an up-convolutional layer (while some later works replace it with a nearest-neighbor up-sampling layer). The last layer of the network performs a $1 \times 1 \times 1$ convolution that resizes the number of output channels to match the number of labels. However, while a couple of 2D images easily fit into a single GPU, a whole 3D images can be too big for the GPU memory, especially for training the network since large memory footprint has to be stored for back-propagation. As one

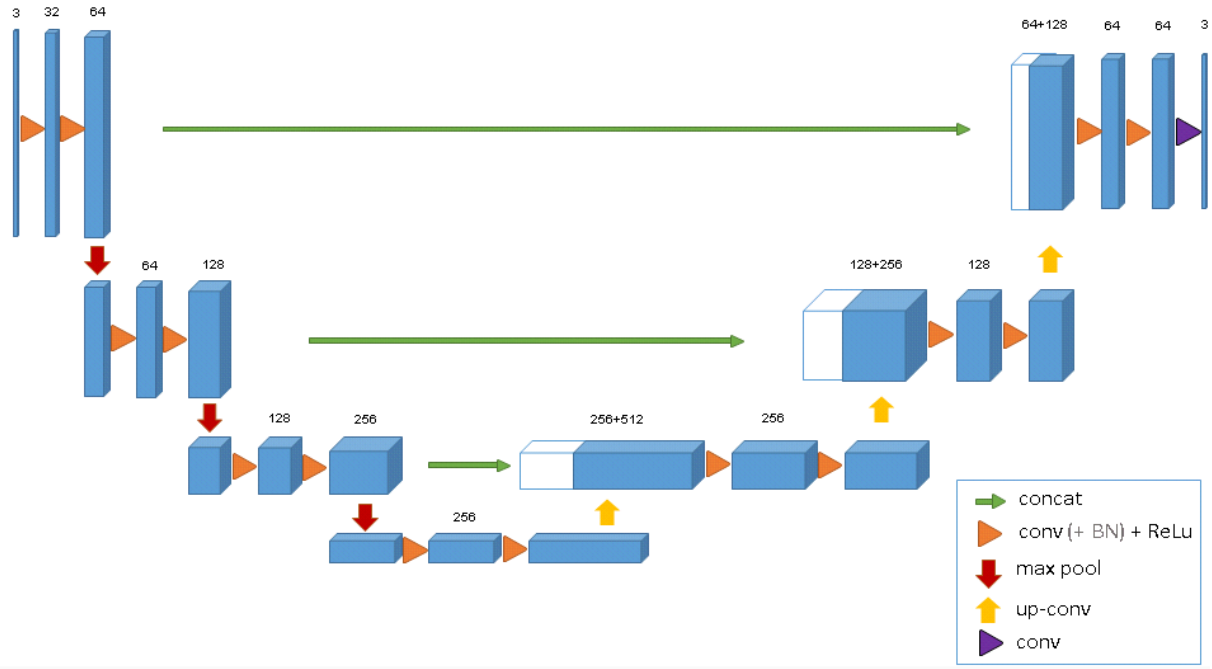


Figure 6.1. The 3D U-Net architecture. Activations are shown as blue cuboids whereas layers and other operations are displayed as arrows. The numbers above cuboids denote the channels of activations.

of the main bottleneck of 3D U-Net, the whole volume sometimes has to be divided into several patches and fed sequentially into the network.

3D U-Net has been serving as a prototype for automatic volumetric segmentation and many later approaches are developed based on the 3D U-Net architecture and modules.

For example, [95] proposes multi-level deep supervision based on the 3D U-Net architecture, in which the three stages in the synthesis path are referred to as three different levels: lower layers, middle layers and upper layers. Besides connecting to the next level, the lower and middle levels (note the upper level is the final stage) are also followed by up-convolutional blocks that upscale their reconstructions to match the input resolution. Therefore, each of the three levels separately produces a segmentation output with the same resolution. It is discussed that the back-propagation performance is improved by calculating losses for the three different outputs, due to the fact that direct supervision on the hidden layers is more effective for the gradient computation.

V-Net [60], which is another volumetric derivation of U-Nets, replaces the pooling layers of the contracting path with 3D convolutions. It is discussed in their paper that convolutions can be applied to reduce the activation resolution by appropriately selecting kernel size and stride, i.e., a kernel size of $2 \times 2 \times 2$ and a stride of 2 halve the resolution of activations. The volumetric convolutional layers increase the receptive field and save the memory footprint during training since they do not need to record the switches that associate the output and input of pooling layers for back-propagation. In addition, each stage (in both the encoder and the decoder) is a residual block in which the input is, after processed by the ReLU non-linearity, added directly to the output of the last convolutional layer. Compared with the non-residual U-Net architecture, the residual modules in V-Net help the network to better converge and achieve higher performance. The very last convolutional layer is similar to that of 3D U-Net, which has a kernel size of $1 \times 1 \times 1$ and it produces a probabilistic segmentation map by applying a voxel-wise softmax function to its output.

In addition, Attention U-Net proposes to highlight the more relevant activations with soft attention modules. To be specific, the authors argue that activations in the synthesis path are relatively imprecise since they are constructed by the up-sampling. Standard U-Nets address the issue with the shortcut paths connecting the analysis path and synthesis path, which, nonetheless, brings heavy redundancy and distracts the network. Therefore, Attention U-net introduces additive soft attention implemented at the shortcut connections on a per-voxel basis, which reduces the computational cost and improve the segmentation performance. Their experiments show that as the number of training epochs increases, Attention U-Net learns to focus more on the foreground areas and they achieve a clear improvement in dice score compared with the standard 3D U-Net.

According to [39], it is commonly believed that more specialized architectures are required for different segmentation tasks and there have been huge amounts of works designed for few or even a single dataset in recent years, which results in troubles for researchers to identify and select the architecture that fits the best in their scenarios. Moreover, those kinds of models generally suffer from overfitting and a lack of adaptation. In this context, [39] proposes nnU-Net with adaptive architectures. In particular, three basic U-Net architectures are included in nnU-Net: 2D-U-Net, 3D-U-Net, and 3D-UNet Cascade, which consists of two 3D-UNet cascaded in sequence to address the memory constraints for large images. All the three architectures are initialized with a specific patch size, batch size, and number of feature maps, which are automatically adjusted according to the median plane size of the training data. A five-fold cross-validation is utilized to choose an architecture (or ensemble) and its topology with the best performance as the final model. Experimental evaluations show that nnU-Net achieves state-of-the-art performance on several distinct datasets and even outperforms the specialized models for some tasks.

While there have been proposing many works on variant specialized architectures for different segmentation applications, they are in generally usually based on the standard 3D U-Net. Therefore, in this paper, we adopt the basic 3D U-Net as our segmentation model with some small modifications to better fit with our problem and approach, which will be explained in the evaluation section.

6.2.2 Network compression and acceleration

On the other hand, as discussed in the previous section, the enormous size and computational cost are currently one of the bottlenecks for these models to be practically deployed. Many methods have been proposed to overcome the efficiency challenge, including quantization [67, 94, 35, 96] [30, 21, 50, 97, 41, 29], pruning, [30] and other encoding approaches [29, 30]. In particular, these works roughly fall into two categories.

The first type of works focus on the on-device storage optimization but gain no computational efficiency improvement to support real-time applications. Although network parameters are compressed into tiny models, they need to be converted back into full-precision values and the computation is carried out using floating-point representation. For example, [30] proposes to “prune” network synapses by forcing some of the weights to zero. In addition, the non-zero weights are clustered into groups and encode the entries using Huffman coding to further reduce the storage per weight. The model can be decoded back into full precision with the code book and they achieve significant compression rate with negligible accuracy loss. The same authors also propose in [97] to quantize weights into ternary values (2-bit weights), which causes very little accuracy degradation by training the quantization centroids. [67] considers the brain segmentation problem and derives their "3DQ" approach based on [97], which also quantizes the full-precision weights into 2 bits. They further incorporate an additional factor to scale the quantization centroids and achieve near full-precision accuracy on two medical imaging 3D segmentation datasets. However, the downside of such technologies is that they do not bring any computational benefits and may even possibly worsen the speed due to the additional decoding phase.

Alternatively, some works directly train the parameters to be integers. In addition to the storage overhead reduction, such approaches also effectively reduce the number of floating-point operation for inferences and improve the computational efficiency. For instance, it is proposed in [94, 35] to operate the neural networks, including training and inferences, with 8-bit-integer weights and activations, where the quantization centroids of [94] is uniformly distributed between -1 and 1, while those of [35] are derived from the maximum absolute values of the weights and activations. Further, DoReFa-Net [96] allows the weights and activations to be quantized into arbitrary bits. They decide the quantization centroids such that the value range of weights is limited to [-1, 1] while activations are bounded within [0, 1]. These works directly approximate the full-precision model with low-bit-width values so that they are able to run with integer arithmetic. On the other hand, some approaches use the low-precision integer to index the quantization centroids. In [41], weights and activations are encoded as non-negative integers on a per-layer basis and can be decoded into full-precision approximations with a pair of shifting and scaling operations. The shifting and scaling factors are directly derived from the full-precision model during the training phase such that all real-valued points fall within the range between the smallest and the greatest quantization centroids, i.e., the clustering is simply performed by taking the full-precision range with a uniformly partition on it. Moreover, they propose a "batch-normalization folding" technique that absorbs the parameters of batch normalization into the previous convolutional or fully-connected layer to reduce the computational complexity.

However, clear accuracy drops are present in the above approaches. The reasons include:

- In [41], an exponential moving average with the smoothing parameter being close to 1 is used to derive The factors for activations. Since the intermediate activations differ from sample to sample, this makes the factors highly depend on the latest batch and relatively volatile.
- Since the weights and activations of a well-trained model mostly follow the Gaussian and half-wave Gaussian distributions [12], respectively, a significant amount of points are concentrating around the mean value and 0. Therefore, for both weights and activations, it is unnecessary and sub-optimal for [35, 41] to span a range covering all samples, especially when using a large mini-batch size or there exist extreme outliers. On the other hand, [94, 96] force the weights between -1 and 1, which as well reduces the performance compared with networks with no such constraints.
- The centroids of weight approximations are not trained in these approaches, but directly computed from the full-precision distributions such that the same ranges are spanned by the quantization centers with the full-precision weights and activations, which makes the accuracy of the full-precision model form an upper bound of the quantized performance. However, due to the definite error introduced by representing continuous ranges using discrete centroids, the drop on performance is inevitable.

The motivation of this work is to improve the previous approaches and address the issues discussed above. In comparison with the first type of works, our approach grants an efficiency improvement on volumetric segmentation with the integer-arithmetic dot-product operations. Moreover, we allow using arbitrary bits for the quantization and aim to reduce the performance degradation by directly training the quantization factors together with other network parameters to minimize the segmentation loss rather than deriving them from the full-precision model, which grants the low-precision model a potential to even outperform the full-precision network.

6.3 Trained affine mapping approach

6.3.1 Learning the mapping of weights

In our approach, each full-precision synaptic weight $\tilde{w}_i$ is encoded as an m-bit integer $g_i \in \mathcal{G}$, where $\mathcal{G} = \{0, 1, \ldots, 2^m - 1\}$ is an affine space. Through a 1D affine mapping, an integer representation can be converted to the following full-precision approximation:

$$\hat{w}_i = S_w(g_i - Z), \tag{6.1}$$

where the linear transformation and the translation are conducted by the scaling factor S_w and the translation factor Z [1], respectively, which are both floating-point numbers. Due to the properties of affine mapping, the uniform codes in $\mathcal{G}$ are mapped to a uniform distribution over the full-precision space, implying our affine mapping approach is essentially a uniform quantizer. However, different from g_i that is non-negative integers, the full-precision centroids span a range over both *positive* and *negative* numbers with the utilization of translation operation. Moreover, by appropriately adjusting the translation factor Z, our approach allows an *asymmetric* partition over the range of positive and negative values. For example, using $m = 2$ bits, $S_w = 0.5$ and $Z = 1$, $g_i = 0, 1, 2, 3$ correspond to the real-valued approximations $\hat{w}_i = -0.5, 0, 0.5, 1$, respectively. This provides us a substantial flexibility in determining and tuning the centroids of our quantizer.

Our training algorithm starts from a pre-trained model, where the full-precision weights practically follow a Gaussian distribution. Figure 6.2 illustrates the weight distribution of a hidden layer in the synthesis path of the pre-trained model, in which μ and σ stand for the

[1] We define S_w and Z on a *per-layer* basis, namely, different pairs of factors are used for different layers. Note this can be easily extended to a *per-kernel* basis to achieve better performance with acceptable additional storage overhead and negligible extra computational cost.

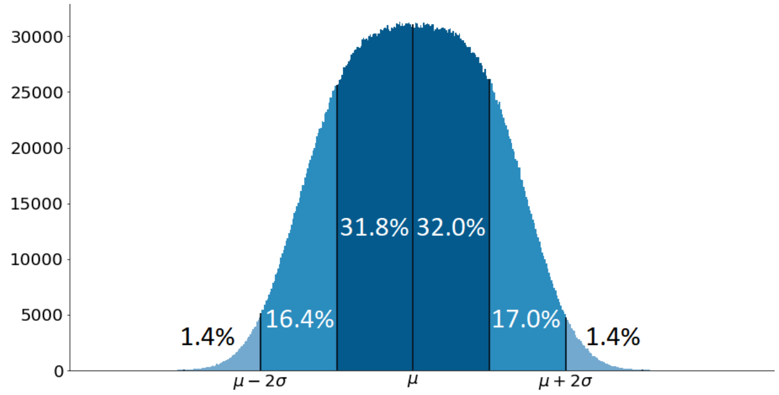


Figure 6.2. The weight distribution of a hidden layer of the pre-trained full-precision model where each band has a width of one standard deviation. The weights generally follow a Gaussian distribution and the values less than one and two standard deviations away from the mean account for 63.8% and 97.2% of the set, respectively.

mean and standard deviation of the distribution, respectively. Empirically, we find that the weight approximation space initialized across the interval of $[\mu - 2\sigma, \mu + 2\sigma]$ leads to a faster convergence and a higher accuracy, compared with the initialization over $[\mu - \sigma, \mu + \sigma]$. This is potentially due to the fact that only around 68% (64% in Figure 6.2) points of a Gaussian distribution fall into the range of $[\mu - \sigma, \mu + \sigma]$. While the rest 32% (36% in Figure 6.2) have relatively large magnitudes and are hence too critical to be clipped. On the other hand, more than 95% of the points are within the range of $[\mu - 2\sigma, \mu + 2\sigma]$ with the rest points tending to be outliers. Thus, clipping the weights beyond $[\mu - 2\sigma, \mu + 2\sigma]$ does not impact the network a lot. In particular, denote the range by $[r_{\min}, r_{\max}]$. We define $r_{\min}$ and $r_{\max}$ as follows:

$$r_{\min} = \mu_{\tilde{W}} - 2\sigma_{\tilde{W}}, \tag{6.2}$$

$$r_{\max} = \mu_{\tilde{W}} + 2\sigma_{\tilde{W}}, \tag{6.3}$$

where $\mu_{\tilde{W}}$ and $\sigma_{\tilde{W}}$ are the per-layer mean and standard deviation of the full-precision weights $\tilde{W}$.

Then, S_w and Z are initialized as follows:

$$S_w = \frac{r_{\max} - r_{\min}}{2^m - 1}, \tag{6.4}$$

$$Z = -\frac{r_{\min}}{S_w}. \tag{6.5}$$

During training, unlike the weight quantization approach in [41] which simply derives their quantization parameters such that the smallest and the greatest centroids equal to the minimal and maximal real-valued weights, the key idea of our approach is re-training the latent full-precision weights to compensate the error introduced by the low-bit-width representation, while the scaling factor S_w and the translation factor Z are concurrently trained against the classification loss independently from other parameters.

During feed-forward pass, each latent full-precision weight $\tilde{w}_i$ is quantized into the low bit-depth code g_i according to:

$$g_i = \text{clip}\left(\text{round}\left(\frac{\tilde{w}_i}{S_w} + Z\right), 0, 2^m - 1\right), \tag{6.6}$$

where

$$\text{clip}\left(x, x_{\min}, x_{\max}\right) = \max\left(x_{\min}, \min(x, x_{\max})\right). \tag{6.7}$$

Then, we use the full-precision approximation expressed in Equation 6.1 to conduct the inference and calculate the loss L.

In back-propagation phase, the latent full-precision weights are injected back in preparation for the update. We use the gradient w.r.t. the weight approximation $\hat{w}_i$ to update the full-precision weight $\tilde{w}_i$:

$$\frac{\partial L}{\partial \tilde{w}_i} = \frac{\partial L}{\partial \hat{w}_i}. \tag{6.8}$$

Additionally, the scaling factor S_w and the translation factor Z are updated concurrently.

Based on chain rule [7], the gradient w.r.t. S_w can be computed from Equation 6.1 as follows:

$$\frac{\partial L}{\partial S_w} = \sum_i \frac{\partial L}{\partial \hat{w}_i} \frac{\partial \hat{w}_i}{\partial S_w} = \sum_i (g_i - Z) \frac{\partial L}{\partial \hat{w}_i}. \tag{6.9}$$

Similarly, we calculate the gradient w.r.t. Z as follows:

$$\frac{\partial L}{\partial Z} = \sum_i \frac{\partial L}{\partial \hat{w}_i} \frac{\partial \hat{w}_i}{\partial Z} = -S_w \sum_i \frac{\partial L}{\partial \hat{w}_i}. \tag{6.10}$$

Then, the latent full-precision weights $\tilde{w}_i$, the scaling factor S_w and the translation factor Z are updated together directly towards the classification loss. As a result, in the next iteration, the full-precision weights $\tilde{w}_i$ and the affine factors have changed, hence the assignment g_i and the approximations $\hat{w}_i$ also have a probability to be different from the previous iteration, which in turn will apply an influence on the gradient w.r.t. S_w and Z in the next back-propagation stage. Generally speaking, our training procedure works essentially in a close manner of relaxation algorithms [56, 63], that repeatedly update both the centroids and the the assignments, while in our problem, the points $\tilde{w}_i$ are moving as well.

6.3.2 Linear mapping of activations

In the previous section, we discussed how full-precision weights can be approximated using m-bit non-negative integers along with an affine mapping operation. Nevertheless, practical implementations also limit the activation precision due to the efficiency challenges. In addition to the weight quantization, we also propose to reduce the activation bit-width based on the half-wave Gaussian quantization (HWGQ) approach proposed in [12].

It is discussed in [12] that batch normalization [38] and relu [28] are widely employed in state-of-the-art CNNs. In particular, the outputs of a convolutional layer are normalized by batch normalization into a Gaussian distribution with zero mean and unit variance. Moreover, relu is a non-linearity that simply drops the negative samples as follows:

$$\phi(x) = \max(0, x), \tag{6.11}$$

and it further trims activations into a half-wave Gaussian distribution.

Based on this observation, given a full-precision activation $\tilde{x}_i$, we would like to encode it with a p-bit unsigned integer $h_i \in \{0, 1, \ldots, 2^p - 1\}$, which corresponds to the floating-point approximation $\hat{x}_i$. Further, since the real-valued activations $\tilde{x}_i$ are half-wave Gaussian distributed with zero mean and unit variance, an optimal quantizer $Q(\tilde{x}_i) = \hat{x}_i$ can be computed by sampling from a standard distribution and applying an iterative relaxation algorithm until convergence. In particular, since the lower bound is explicitly defined at 0, which is also the crest of the distribution, we drop the translation term and approximate activations with the following linear mapping function:

$$\hat{x}_i = Q(\tilde{x}_i) = S_a h_i, \tag{6.12}$$

where S_a is a linear scaling factor of floating-point value, and the assignment variable h_i can be derived from $\tilde{x}_i$ as follows:

$$h_i = \text{clip}\left(\text{round}\left(\frac{\tilde{x}_i}{S_a}\right), 0, 2^p - 1\right). \tag{6.13}$$

For the scaling factor S_a, recall that batch normalization generally produces response activations that are Gaussianly distributed with zero mean and unit variance across all layers. Therefore, given the quantization precision p, there is no need to define or train different factors for different layers, and a same linear quantizer can be used across the network. In particular, an optimal quantizer defined on a distribution can be expressed in the sense of quadratic error minimization:

$$\arg\min_{Q} \int \varphi(x)(Q(x) - x)^2 dx. \tag{6.14}$$

Although Lloyd's algorithm [56] can be generally applied to solve the clustering problems, it breaks the linear constraints on centroids. Therefore, we propose a variation of Lloyd's algorithm to overcome this issue. During the step for center update, instead of computing the new centroids by simply taking an average for each cluster, all points are first normalized on a per-cluster basis such that they are on the magnitude of one scaling factor, and then the mean value is derived from all normalized samples as the updated factor S_a. In other words, we update S_a as follows:

$$S_a = \frac{\sum_{i \,\text{for}\, h_i \neq 0} \frac{\tilde{x}_i}{h_i}}{\sum_{i \,\text{for}\, h_i \neq 0} 1}. \tag{6.15}$$

A brief description of our clustering algorithm is summarized in Algorithm 4.

Algorithm 4. Computation of activation scaling factor S_a

Input: $p, \tilde{X} = \{\tilde{x}_i\}$ sampled from a standard half-wave Gaussian distribution
Output: S_a

```
while $H$ not converged do
    step 1: update centroids $\{0, S_a, \ldots, (2^p - 1)S_a\}$
    step 2: update $H$ according to Equation 6.13
    step 3: update $S_a$ according to Equation 6.15
end while
return $\tilde{\pi}$
```

In feed-forward pass, the real-valued activations $\tilde{x}_i$ are quantized into the floating-point approximations $\hat{x}_i$. However, another problem is introduced by the quantization of activations, that during back-propagation phase, the stair-like rounding operation in Equation 6.13 makes the function completely non-differentiable and breaks the gradient chain. To address this issue, rather than propagating gradient back from the approximations $\hat{x}_i$ to the full-precision outputs $\tilde{x}_i$, we explicitly define that the real-valued activations directly inherit the derivatives from their approximations and skip computing the derivatives of the non-differentiable rounding operations.

Moreover, it is discussed in [12] that since the activations with extremely large values are bounded to the greatest centroid by quantization, it causes a problem of gradient mismatch [52] by deriving derivatives from the quantized results. Therefore, we adopt the gradient clipping scheme proposed in [66] and discard the derivatives of the real-valued activations that are beyond the range of our quantization centroids. To be specific, the gradient w.r.t. the activations before quantization is computed as follows:

$$\frac{\partial L}{\partial \tilde{x}_i} = \frac{\partial L}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial \tilde{x}_i}, \tag{6.16}$$

where $\frac{\partial \hat{x}_i}{\partial \tilde{x}_i}$ is illustrated in Figure 6.3.

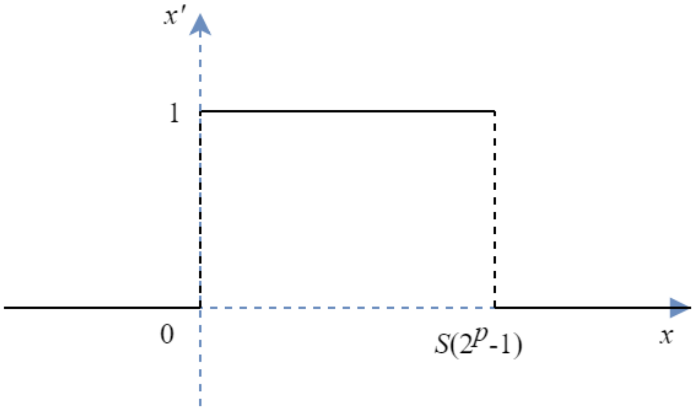


Figure 6.3. The gradient of approximations w.r.t. the real-valued activations based on the gradient clipping approach.

6.3.3 Optimizing 3D U-Net based Brain Tumor Segmentation

As discussed above, our proposed approach encodes weights and activations with m-bit and p-bit unsigned integers, respectively. However, it can be seen in Equations 6.1 and 6.12 that, the approximations after decoding are still carried out in a floating-point format, which do not effectively reduce the computational power. In this section, we study that after training, how inference can be operated in a more efficient manner. Specifically, the inference of a neuron followed by batch normalization and relu can be straightforwardly implemented as follows:

1. Activation decoding: $\hat{x}_i = S_a h_i$;

2. Weight decoding: $\hat{w}_i = S_w(g_i - Z)$;

3. Dot-product: $\tilde{y} = \sum_i \hat{w}_i \hat{x}_i + b$;

4. Batch normalization & relu: $\tilde{z} = \max\left(0, \frac{\tilde{y}-\mu}{\sqrt{\sigma^2+\varepsilon}}\right)$;

5. Activation encoding: $h_{\text{out}} = \text{clip}\left(\text{round}\left(\frac{\tilde{z}}{S_a}\right), 0, 2^p - 1\right)$;

where b is the bias of convolutional layers.

By combining steps 1, 2, and 3, the intermediate output $\tilde{y}$ can be simplified as follows:

$$\tilde{y} = S_w S_a \sum_i g_i h_i - S_w S_a Z \sum_i h_i + b. \tag{6.17}$$

As defined in the previous sections, g_i and h_i are non-negative integers of m and p bits, respectively. Therefore, the *dot-product* operations can be essentially performed using *integer-only arithmetic* with the floating-point multiplications deferred after that. Furthermore, parameters in steps 4, 5 and Equation 6.17 become constants once the training is complete, hence can be absorbed into a single 2D affine function along with the rounding and clipping non-linearity:

$$h_{\text{out}} = \text{clip}\left(\text{round}\left(Av_1 + Bv_2 + C\right), 0, 2^p - 1\right), \tag{6.18}$$

where

$$v_1 = \sum_i g_i h_i, \tag{6.19}$$

$$v_2 = \sum_i h_i, \tag{6.20}$$

$$A = \frac{S_w}{\sqrt{\sigma^2+\varepsilon}}, \tag{6.21}$$

$$B = -\frac{S_w Z}{\sqrt{\sigma^2+\varepsilon}}, \tag{6.22}$$

$$C = \frac{b-\mu}{S_a\sqrt{\sigma^2+\varepsilon}}. \tag{6.23}$$

In conclusion, A, B and C can be pre-computed and the inference procedure reduces into two steps:

1. Compute v_1 and v_2 according to Equations 6.19 and 6.20 in the integer domain;

2. Compute h_{out} according to Equation 6.18 in the floating-point domain.

Note that the floating-point operations in step 2 are conducted on a per-neuron basis, which is not in domination of the computational complexity. In this way, our trained affine mapping approach efficiently produces an improvement in terms of both storage and computational cost.

Moreover, some network architectures employ shortcut connections that concatenate the outputs of two layers. For example, in regards of 3D U-Net, the output at each stage of the encoder is directly concatenated with the input of the decoder at the same stage. However, since the same activation scaling factor S_a is adopted throughout all layers, a full-precision activation $\tilde{z}$ shall be encoded to the same unsigned integer h_{out} regardless of which layer it belongs to. Therefore, stacking the intermediate activation $\tilde{z}$ in step 4 is essentially equivalent to stacking the encoded value h_{out} in step 5, which leads to no additional operations besides independently computing h_{out} for the two layers according to Equation 6.18.

6.4 Evaluation

6.4.1 Experimental setup

We evaluate our approach on the BraTS 2018 challenge discussed in Section 6.1. In particular, while the ground truth segmentation of the official BraTS 2018 validation dataset is not publicly available, to perform hyper-parameter tuning and provide clear comparison between the prediction and ground truth, in spite of the official validation dataset, we use 10% of the training dataset as our validation dataset in our experiments and train the models with the rest 90% samples[2]. The N4ITK bias correction [88] approach is applied to all the MRI images to

[2] Note that our experiments aim to illustrate the difference in performance between the full-precision and our quantized models. The full-precision model serves as a baseline and its absolute performance is not critical.

reduce the bias caused by using different scanners. We then clip the greatest as well as the smallest 2% voxels in each channel (modality) to remove outliers. Lastly, images are normalized to zero mean and unit variance on a per-channel basis while the non-brain regions are set to 0.

We employ the minorly modified 3D U-Net [99] in our experiments. In particular, we set the number of input channels to 4 to match the 4 modalities of the MRI scans. Moreover, we follow the approach of the 1st-place winner of BraTS 2018 [62], where the output of the network has 3 channels corresponding to the 3 tumor sub-region labels, i.e., the whole tumor (WT), the tumor core (TC), and the enhancing tumor (ET), and they are then connected to a sigmoid activation function that produces the predicted probabilities of each label. As described in [99], batch-normalization is introduced before each ReLU non-linearity, which is also adopted in our implementation so the architecture is compatible with our activation quantization scheme as discussed in Section 6.3.2. The up-convolutional layers in the original architecture are replaced with up-sampling layers (Note that while we choose to use up-sampling layers, it as well fits with our approach if applying batch-normalization and ReLU after each up-convolutional layer). We use the multi-class dice loss function based on the dice loss proposed in [60]. According to [60], denote by p_i and q_i the voxels at the same location in the prediction and ground truth for a class, respectively, the single-class dice loss is defined as follows:

$$L_{\text{class}} = -\frac{2\sum_i p_i q_i}{\sum_i p_i^2 + \sum_i q_i^2}. \tag{6.24}$$

Given the dice loss of three tumor sub-regions L_{WT}, L_{TC}, and L_{ET}, our loss function is simply the summation of these three scores as follows:

$$L = L_{\text{WT}} + L_{\text{TC}} + L_{\text{ET}}. \tag{6.25}$$

In other words, the three tumor sub-regions are assigned with identical weights.

We randomly sample patches of size $96 \times 96 \times 96$ with a batch size of 2 to train the networks. All the low-precision models start from the pre-trained full-precision model and we re-train them using our quantization algorithm. We adopt an Adam optimizer to update the scaling and translation factors S and Z, with the learning rate initialized to be 1e-6 (While the gradient w.r.t. these factors is accumulated by every weight and the amount of weights in a 3D convolutional layer can be significant, it empirically works better with relatively small learning rates). The other parameters (weights and biases, etc.) are tuned by another Adam optimizer with a learning rate of 1e-4 and a weight decay of 1e-5. We divide the learning rates of both optimizer by 10 every 1,000 batches and the models are trained for 4,000 batches.

The validation images are padded with 0 and partitioned into multiple $96 \times 96 \times 96$ patches. We feed the network to produce predictions of the same shape and reconstruct the label map. Different from the training procedure where we use probabilistic output to compute the dice loss, in evaluation we binarize the output (before the sigmoid) with a threshold of 0 and used the binarized reconstruction to compare with ground truth and compute dice scores.

6.4.2 Segmentation results

Table 6.1. Validation results (mean dice scores) on the BraTS 2018 dataset. W bits and A bits represent the number of bits used to encode weights and activations. WT, TC, and ET stand for the whole tumor, the tumor core, and the enhancing tumor, respectively. Experiments are repeated three times and we report the average results for validation, while standard deviations are shown in parenthesis.

	W bits	A bits	WT	TC	ET
FP			0.888 (0.0006)	0.801 (0.0073)	0.762 (0.0029)
FPN	8	8	0.744 (0.0456)	0.361 (0.1542)	0.144 (0.1786)
	4	4	0.666 (0.0921)	0.189 (0.1313)	0.074 (0.1039)
	2	2	0 (0)	0 (0)	0 (0)
3DQ	2	float	0.885 (0.0026)	0.792 (0.0014)	0.747 (0.0029)
DoReFa	8	8	0.874 (0.0036)	0.796 (0.0047)	0.754 (0.0047)
	4	4	0.873 (0.0017)	0.789 (0.0018)	0.750 (0.0022)
	2	2	0.874 (0.0024)	0.787 (0.0032)	0.750 (0.0038)
Ours	8	8	0.887 (0.0019)	0.801 (0.0020)	0.767 (0.0021)
	4	4	0.889 (0.0028)	0.797 (0.0016)	0.761 (0.0006)
	2	2	0.884 (0.0014)	0.793 (0.0014)	0.760 (0.0006)

Besides our trained affine mapping (TAM) approach, we also evaluate the full-precision (FP) model along with three low-precision CNN training and inference techniques as our baselines: 3DQ [67], DoReFa-Net [96], and the naïve Fixed-Point Number (FPN) representation [65]. The first two techniques are previously discussed in Section 6.2, and FPN is the simplest quantization approach, which allows the valid bits to represent any continuous power-of-2 fractional values. For example, the 2-bit FPN 1.1, which consists of an integer bit and a fractional bit, encodes the decimal value $1 \times 2^0 + 1 \times 2^{-1} = 1.5$. In our FPN experiments, the weights are quantized using an additional bit as the sign bit, and we allow the represented bits (as long as they are continuous) to be away from the radix point while skipping the other more significant digits, e.g., using 3 unsigned bits 101 as in 0.0101 to encode $1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0.3125$. In particular, all low-precision approaches except for FPN are re-trained from a pre-trained full-precision model, which is also used to derive the quantized weights for FPN so that the represented bits minimize the mean squared error. In addition, since the activations are dependant on the input, it does not make sense for FPN to dynamically compute the best scales, thus we simply truncate the activations with all bits being fractional bits. All models are trained on a NVIDIA RTX 2080 Ti GPU and we compute the dice coefficients for the three tumor sub-regions. All experiments are repeated three times and the average results are summarized in Table 6.1.

Among all low-precision techniques, FPN achives the worst performance. Even when using 8-bit weights and activations, the average dice for WT drastically drops from 0.888 to 0.744, while the TC and ET dice scores fall below 0.4 and 0.2, respectively. The performance further becomes worse with 4-bit weights and activations, and the model totally does not produce any useful information with 2-bit precision, which gets the dice coefficients of 0 for all the three sub-regions. The reason is that the error caused by such post-training compression approaches is not compensated so the error accumulates throughout layers and critically hurts the model.

On the other hand, our trained affine mapping approach with 8-bit weights and activations achieves 0.887, 0.801, and 0.767 average dice for WT, TC, and ET, respectively, which are close to (or even better than) the full-precision model, while there is a dice loss of about 0.01 present in the 8-bit DoReFa-Net results. Moreover, TAM achieves negligibly small loss with 4-bit weights and activations (within 0.005 of the full-precision model), and about 0.01 degradation when using 2-bit precision, whereas DoReFa-Net shows a clearer performance drop in such cases. 3DQ achieves fairly good WT score, but it performs poorly for the TC and ET sub-regions. We also note that 3DQ uses floating-point activations and it is a compression approach that does not accelerate the inference, which is not as beneficial other baselines. The validation results, i.e., our approach outperforms other baselines and achieves a performance close to the full-precision model, verify that our factor-training scheme effectively tunes the centroids and reduces the loss introduced by the quantization.

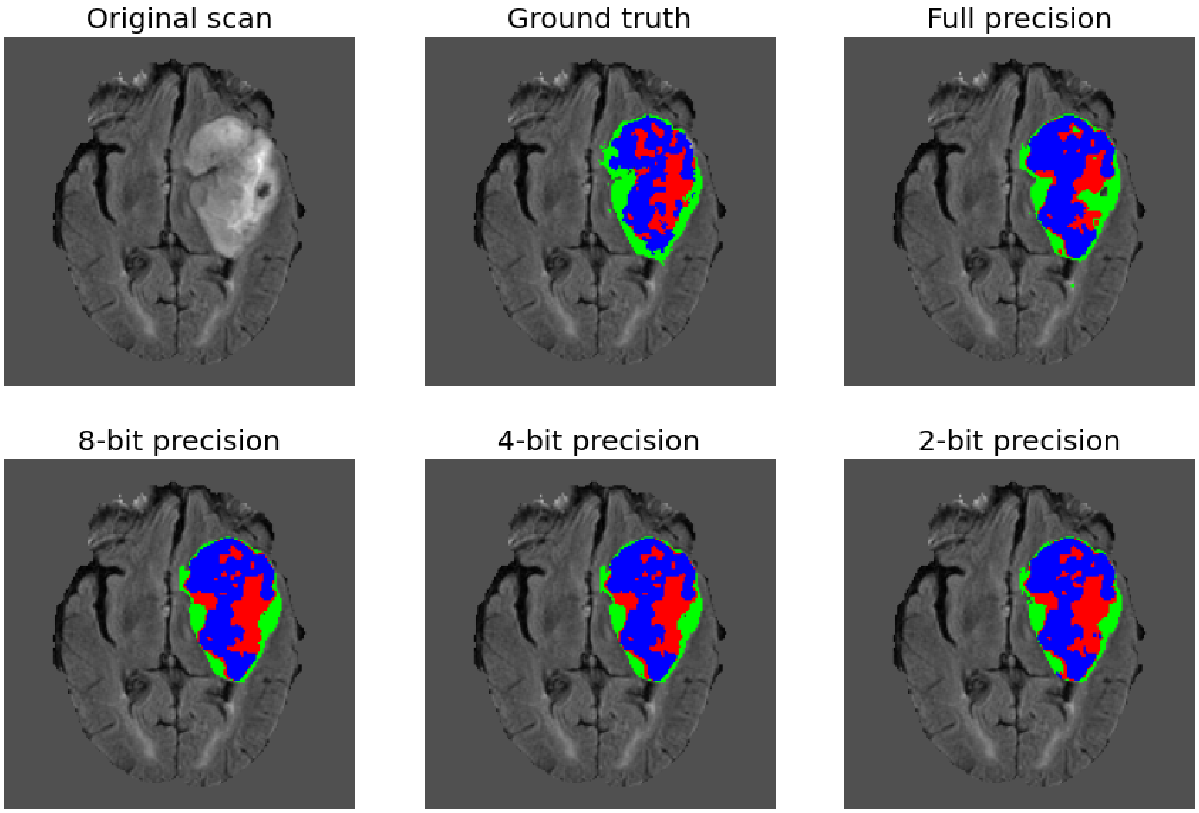


Figure 6.4. A example of the predictions of the full-precision and our quantized models along with the ground truth annotations overlaid over the FLAIR MRI scan. The necrotic and non-enhancing tumor core (NCR & NET) are shown in red; the peritumoral edema (ED) is shown in green; and the GD-enhancing tumor (ET) is in shown in blue. The prediction label WT is the combination of all the three colored areas and TC is the union of red and blue.

In addition, we reconstruct the different parts of tumors (NCR & NET, ED, and ET) from our predicted labels to qualitatively illustrate our results. An axially sliced example from our validation set with the FLAIR modality as background is presented in Figure 6.4. As can be observed in the ground truth annotation, the red area (NCR & NET) has relatively more irregular shape, which makes it the most difficult part to accurately predict, while the green (ED) and blue (ET) regions also have some dotted details around their boundaries, i.e., some blue dots in the green area and green dots outside the main tumor. However, while all models perform badly in predicting the red region, the full-precision model additionally tries to capture the dotted feature of the ground truth, potentially due to the over-fitting problem though regularization is already applied during training. However, it is almost impossibly to perfectly predict these small dots and hence this actually increases the error of the full-precision model. Nevertheless, our quantized models produce relatively smoother annotations without these small dots. This might explain the reason why our quantized models sometimes outperform the full-precision model.

6.5 Conclusion

In this paper, we consider the problem of optimizing the 3D U-Net with low-precision parameters and integer-arithmetic inference for efficient volumetric segmentation. In particular, we propose a trained affine mapping approach that encodes weights and activations as non-negative integers of dedicated bit-widths, and recovers the floating-point approximations with affine mapping functions. The key idea of our work is that the scaling and translation factors for weights can be trained together with other parameters, whereas activations are generally normalized by batch normalization and rectified linear units (ReLU), hence can be accurately approximated using the same function across all layers, which is pre-computed based on the standard half-wave Gaussian distribution. In addition, with weights and activaions encoded as low-precision integers, we propose to defer the floating-point computation of the affine mapping functions and combine it with the quantization procedure of the next layer. This technique simplifies the inference into two steps, in which the dot-product operations are carried out using unsigned-integer arithmetic and the floating-point multiplications are reduced onto a per-neuron basis. Evaluation results on the BraTS 2018 challenge show that the models quantized by our trained affine mapping algorithm using 2-bit weights and activations achieve a mean dice score within 0.01 relative to the full-precision model. Furthermore, our quantization achieves negligibly small degradation with 4-bit and 8-bit precisions.

6.6 Acknowledgements

Chapter 6, in full, is a reprint of the material as it appears in ACM Journal on Emerging Technologies in Computing Systems, 2022. Weijia Wang and Bill Lin. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[2] Sercan O Arık and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv*, 2020.

[3] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques, sept 2019.

[4] Gilles Audemard, Frédéric Koriche, and Pierre Marquis. On Tractable XAI Queries based on Compiled Representations. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 838–849, 9 2020.

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

[6] Dimitris Bertsimas, Allison Chang, and Cynthia Rudin. Orc: Ordered rules for classification a discrete optimization approach to associative classification. 09 2012.

[7] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.

[8] E. Boros, P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*,

12(2):292–306, 2000.

[9] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[10] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[11] G. W. Burr, P. Narayanan, R. M. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici. Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power). In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 4.4.1–4.4.4, Dec 2015.

[12] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. *CoRR*, abs/1702.00953, 2017.

[13] E. Candès, M. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted l1 minimization. *Journal of Fourier Analysis and Applications*, 14:877–905, 2007.

[14] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. An interpretable model with globally consistent explanations for credit risk, 2018.

[15] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[16] Z. Chen, B. Gao, Z. Zhou, P. Huang, H. Li, W. Ma, D. Zhu, L. Liu, X. Liu, J. Kang, and H. Y. Chen. Optimized learning scheme for grayscale image recognition in a rram based analog neuromorphic system. In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 17.7.1–17.7.4, Dec 2015.

[17] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–39, June 2016.

[18] Arthur Choi, Weijia Shi, Andy Shih, and Adnan Darwiche. Compiling neural networks into tractable boolean circuits. *intelligence*, 2017.

[19] Peter Clark and Tim Niblett. The cn2 induction algorithm. *Mach. Learn.*, 3(4):261–283, March 1989.

[20] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[21] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014.

[23] Y. Crama, P. L. Hammer, and T. Ibaraki. Cause-effect relationships and partially defined boolean functions. *Ann. Oper. Res.*, 16(1–4):299–325, January 1988.

[24] S. Dash, D. M. Malioutov, and K. R. Varshney. Screening for learning classification rules via boolean compressed sensing. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3360–3364, 2014.

[25] Sanjeeb Dash, Oktay Günlük, and Dennis Wei. Boolean decision rules via column generation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4660–4670, Red Hook, NY, USA, 2018. Curran Associates Inc.

[26] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, dec 2006.

[27] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[29] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

[30] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[31] Song Han, Jeff Pool, John Tran, and W. Dally. Learning both weights and connections for efficient neural network. *ArXiv*, abs/1506.02626, 2015.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[33] M. Hu, H. Li, Q. Wu, and G. S. Rose. Hardware realization of bsb recall function using memristor crossbar arrays. In *DAC Design Automation Conference 2012*, pages 498–503,

June 2012.

[34] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[35] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016.

[36] Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. *CoRR*, abs/1811.10656, 2018.

[37] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.

[39] Fabian Isensee, Jens Petersen, André Klein, David Zimmerer, Paul F. Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Köhler, Tobias Norajitra, Sebastian J. Wirkert, and Klaus H. Maier-Hein. nnu-net: Self-adapting framework for u-net-based medical image segmentation. *CoRR*, abs/1809.10486, 2018.

[40] Yacine Izza and João Marques-Silva. On explaining random forests with SAT. *CoRR*, abs/2105.10278, 2021.

[41] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017.

[42] Yu Ji, Youhui Zhang, Wenguang Chen, and Yuan Xie. Bridging the gap between neural networks and neuromorphic hardware with A neural network compiler. *CoRR*, abs/1801.00746, 2018.

[43] Liran Katzir, Gal Elidan, and Ran El-Yaniv. Net-dnf: Effective deep modeling of tabular data. In *International Conference on Learning Representations*, 2020.

[44] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[45] Yongtae Kim, Yong Zhang, and Peng Li. A reconfigurable digital neuromorphic processor

with memristive synaptic crossbar for cognitive computing. *J. Emerg. Technol. Comput. Syst.*, 11(4):38:1–38:25, April 2015.

[46] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency, 2017.

[47] 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*, KDD '16, page 1675–1684, New York, NY, USA, 2016. Association for Computing Machinery.

[48] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model, 2015.

[49] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang. Memristor-based approximated computation. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 242–247, Sept 2013.

[50] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

[51] Yang Li and Shihao Ji. l_0-arm: Network sparsification via stochastic binary optimization, 2019.

[52] Darryl Dexu Lin and Sachin S. Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *CoRR*, abs/1607.02241, 2016.

[53] Bill Lin Litao Qiao, Weijia Wang. Learning accurate and interpretable decision rule sets from neural networks. In *35th AAAI Conference on Artificial Intelligence*, 2021.

[54] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, page 80–86. AAAI Press, 1998.

[55] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Yu Wang, Hao Jiang, M. Barnell, Qing Wu, and Jianhua Yang. Reno: A high-efficient reconfigurable neuromorphic computing accelerator design. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[56] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[57] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for

semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[58] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2017.

[59] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, 2015.

[60] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation, 2016.

[61] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *CoRR*, abs/1603.01025, 2016.

[62] Andriy Myronenko. 3d mri brain tumor segmentation using autoencoder regularization, 2018.

[63] S. Na, L. Xumin, and G. Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 63–67, April 2010.

[64] Nvidia. Nvidia deep learning accelerator, 2018.

[65] Erick L Oberstar. Fixed-point representation & fractional math. *Oberstar Consulting*, 9, 2007.

[66] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[67] Magdalini Paschali, Stefano Gasperini, Abhijit Guha Roy, Michael Y.-S. Fang, and Nassir Navab. 3dq: Compact quantized neural networks for volumetric whole brain segmentation. *CoRR*, abs/1904.03110, 2019.

[68] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differ-

entiation in pytorch. In *NIPS-W*, 2017.

[69] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[70] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521:61 EP –, May 2015.

[71] Litao Qiao, Weijia Wang, and Bill Lin. Learning accurate and interpretable decision rule sets from neural networks. In *AAAI*, 2021.

[72] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should i trust you?”: Explaining the predictions of any classifier, 2016.

[73] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, 2018.

[74] Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, November 1987.

[75] Lior Rokach and Oded Maimon. Top–down induction of decision trees classifiers–a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, page 487, 2005.

[76] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

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

[78] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML ’97, page 313–321, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[79] Peter Schmidt and Ann D Witte. *Predicting recidivism in north carolina, 1978 and 1980*. Inter-university Consortium for Political and Social Research, 1988.

[80] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, June 2016.

[81] Weijia Shi, Andy Shih, Adnan Darwiche, and Arthur Choi. On tractable representations of binary neural networks, 2020.

[82] Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers, 2018.

[83] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[84] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[85] L. Song, X. Qian, H. Li, and Y. Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, Feb 2017.

[86] Guolong Su, Dennis Wei, Kush R. Varshney, and Dmitry M. Malioutov. Interpretable two-level boolean rule learning for classification, 2015.

[87] T. M. Taha, R. Hasan, C. Yakopcic, and M. R. McLean. Exploring the design space of specialized multicore neural processors. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013.

[88] N. J. Tustison, B. B. Avants, P. A. Cook, Y. Zheng, A. Egan, P. A. Yushkevich, and J. C. Gee. N4itk: Improved n3 bias correction. *IEEE Transactions on Medical Imaging*, 29(6):1310–1320, 2010.

[89] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[90] Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. 2018.

[91] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. A bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 18(70):1–37, 2017.

[92] H.-S. Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T. Chen, and Ming-Jinn Tsai. Metal–oxide rram. *Proceedings of the IEEE*, 100(6):1951–1970, 2012.

[93] C. Yakopcic and T. M. Taha. Energy efficient perceptron pattern recognition using seg-

mented memristor crossbar arrays. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013.

[94] Yukuan Yang, Shuang Wu, Lei Deng, Tianyi Yan, Yuan Xie, and Guoqi Li. Training high-performance and large-scale deep neural networks with full 8-bit integers. *CoRR*, abs/1909.02384, 2019.

[95] Guodong Zeng, Xin Yang, Jing Li, Lequan Yu, Pheng-Ann Heng, and Guoyan Zheng. 3d u-net with multi-level deep supervision: Fully automatic segmentation of proximal femur in 3d mr images. pages 274–282, 09 2017.

[96] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

[97] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.

[98] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian D. Reid. Towards effective low-bitwidth convolutional neural networks. *CoRR*, abs/1711.00205, 2017.

[99] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation, 2016.