Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Special Session on KR and Machine Learning

On Tractable Representations of Binary Neural Networks

Weijia Shi , Andy Shih , Adnan Darwiche , Arthur Choi
Computer Science Department, University of California, Los Angeles, California, USA
{swj0419,andyshih,darwiche,aychoi}@cs.ucla.edu

Abstract

We consider the compilation of a binary neural network's decision function into tractable representations such as Ordered Binary Decision Diagrams (OBDDs) and Sentential Decision Diagrams (SDDs). Obtaining this function as an OBDD/SDD facilitates the explanation and formal verification of a neural network's behavior. First, we consider the task of verifying the robustness of a neural network, and show how we can compute the expected robustness of a neural network, given an OBDD/SDD representation of it. Next, we consider a more efficient approach for compiling neural networks, based on a pseudo-polynomial time algorithm for compiling a neuron. We then provide a case study in a handwritten digits dataset, highlighting how two neural networks trained from the same dataset can have very high accuracies, yet have very different levels of robustness. Finally, in experiments, we show that it is feasible to obtain compact representations of neural networks as SDDs.

1 Introduction

Recent progress in artificial intelligence and the increased deployment of AI systems have highlighted the need for *explaining* the decisions made by such systems; see, e.g., (Baehrens et al. 2010; Ribeiro, Singh, and Guestrin 2016b; Ribeiro, Singh, and Guestrin 2018; Lipton 2018; Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019b; Darwiche and Hirth 2020).[1] For example, one may want to explain *why* a classifier decided to turn down a loan application, or rejected an applicant for an academic program, or recommended surgery for a patient. Answering such *why?* questions is particularly central to assigning blame and responsibility, which lies at the heart of legal systems and is further a requirement in certain contexts.[2] The *formal verification* of AI systems has also come into focus recently, particularly when such systems are deployed in safety-critical applications.

We propose a *knowledge compilation* approach for explaining and verifying the behavior of a neural network classifier. Knowledge compilation is a sub-field of AI that studies in part *tractable* Boolean circuits, and the trade-offs between succinctness and tractability (Selman and Kautz 1996; Cadoli and Donini 1997; Darwiche and Marquis 2002; Darwiche 2014). By enforcing different properties on the structure of a Boolean circuit, one can obtain greater tractability (the ability to perform certain queries and transformations in polytime) at the possible expense of succinctness (the size of the resulting circuits). Our goal is to compile the Boolean function specified by a neural network into a tractable Boolean circuit that facilitates explanation and verification.

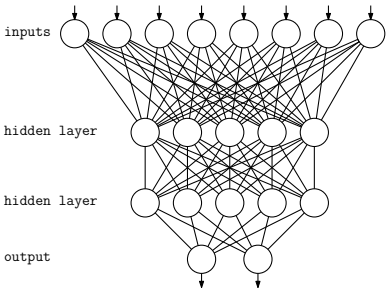
We consider neural networks whose inputs are binary $(0/1)$ and that use step activations. Such a network would have real-valued parameters, but the network itself induces a purely Boolean function. We seek a *tractable Boolean circuit* that represents this function, which we obtain in two steps. First, note that neurons with step activations and binary inputs then produce a binary output—each neuron induces its own Boolean function. Using, e.g., the algorithm of (Chan and Darwiche 2003) we can obtain a tractable circuit for a given neuron's Boolean function. The neural network then induces a Boolean circuit, although it may not be tractable. Thus, we *compile* this circuit into a tractable one by enforcing additional properties on the circuit until certain operations become tractable, as done in the field of knowledge compilation. We then explain the decisions and verify the properties of this circuit, as done in (Shih, Choi, and Darwiche 2018b; Shih, Choi, and Darwiche 2018a); cf. (Darwiche and Hirth 2020).

Our approach follows a recent trend in analyzing machine learning models using symbolic approaches such as satisfiability and satisfiability modulo theory; see, e.g., (Katz et al. 2017; Leofante et al. 2018; Narodytska et al. 2018; Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019a; Ignatiev, Narodytska, and Marques-Silva 2019b; Shih, Darwiche, and Choi 2019; Audemard, Koriche, and Marquis 2020). While machine learning and statistical methods are key for learning classifiers, it is clear that symbolic and logical approaches, which are independent of any of the models parameters, are key for analyzing and reasoning about them. Our approach, based on compilation into a tractable Boolean circuit, can go beyond queries based on (for example) satisfiability, as we shall show.

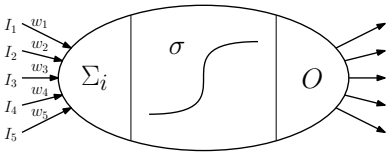
This paper is organized as follows. In Section 2, we re-

[1] It is now recognized that opacity, or lack of explainability is "one of the biggest obstacles to widespread adoption of artificial intelligence" (The Wall Street Journal, August 10, 2017).

[2] Take for example the European Union general data protection regulation, which has a provision relating to explainability, http://www.privacy-regulation.eu/en/r71.htm.



(a) A neural network structure



(b) A mathematical model of a neuron

Figure 1: A neural network and a neuron. A sigmoid activation $\sigma(x) = \frac{1}{1+\exp\{-x\}}$ acts as a soft threshold which tends to 0 and 1 as x goes to $-\infty$ and ∞, respectively. A ReLU activation $\sigma(x) = \max(0, x)$ outputs 0 if $x < 0$ and outputs x otherwise.

view relevant background material. In Section 3, we show how to reduce neural networks to Boolean circuits by compiling each neuron into a Boolean circuit. In Section 4, we discuss how to obtain tractable circuits, via knowledge compilation. In Section 5, we show how a tractable circuit enables one to reason about the robustness of a neural network. In Section 6, we provide a case study with experimental results and finally conclude with a discussion in Section 7.

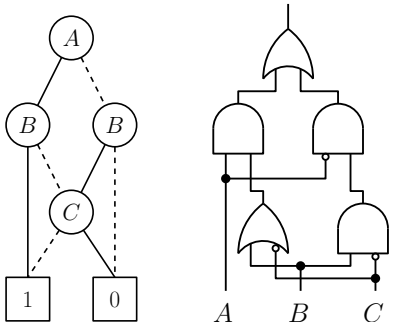
2 Technical Preliminaries

A feedforward neural network is a directed acyclic graph (DAG); see Figure 1a. The roots of the DAG are the neural network inputs, call them $X_1, \ldots, X_n$. The leaves of the DAG are the neural network outputs, call them $Y_1, \ldots, Y_m$. Each node in the DAG is called a *neuron* and contains an *activation function* σ; see Figure 1b. Each edge I in the DAG has a *weight* w attached to it. The weights of a neural network are its *parameters,* which are learned from data.

In this paper, we assume that the network inputs X_i are either 0 or 1. We further assume *step* activation functions:

$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

A neuron with a step activation function has outputs that are also 0 or 1. If the network inputs are also 0 or 1, then this means that the inputs to all neurons are 0 or 1. Moreover, the output of the neural network is also 0 or 1. Hence, each neuron and the network itself can be viewed as a function mapping binary inputs to a binary output, i.e., a Boolean function. For each neuron, we shall simply refer to this function as the *neuron's Boolean function*. When there is a single output Y, we will simply refer to the corresponding function as the *network's Boolean function*.



(a) An OBDD (b) A circuit

Figure 2: An OBDD and circuit representation of a neuron $\sigma(A + B - C - 1)$ where σ is a step activation function.

3 From Neural Networks to Boolean Circuits

Consider a neuron with step activation function σ, inputs I_i, weights w_i and bias b. The output of this neuron is simply

$$\sigma(\sum_i w_i \cdot I_i + b) = \begin{cases} 1 & \text{if } \sum_i w_i \cdot I_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

As an example, consider a neuron with 3 inputs A, B and C with weights $w_1 = 1.15, w_2 = 0.95$ and $w_3 = -1.05$ and a bias of -0.52. This neuron outputs 1 iff:

$$1.15 \cdot A + 0.95 \cdot B - 1.05 \cdot C \geq 0.52$$

Treating a value of 1 as true and a value of 0 as false, we can view this neuron as a Boolean function $f(A, B, C)$ whose output matches that of the neuron, on inputs A, B and C. Figure 2 highlights two logically equivalent representations of this neuron's Boolean function. Figure 2a highlights an Ordered Binary Decision Diagram (OBDD) representation[3] and Figure 2b highlights a circuit representation. These functions are equivalent to the sentence:

$$[\neg C \wedge (A \vee B)] \vee [C \wedge A \wedge B],$$

i.e., if C is 0 then A or B must be 1 to meet or surpass the threshold (≥ 0), and if C is 1 then both A and B must be 1.

OBDDs, as in Figure 2a, are *tractable* representations—they support many operations in time *polynomial* (and typically *linear*) in the size of the OBDD (Bryant 1986; Meinel

[3] An Ordered Binary Decision Diagram (OBDD) is a rooted DAG with two sinks: a 1-sink and a 0-sink. An OBDD is a graphical representation of a Boolean function on variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Every OBDD node (but the sinks) is labeled with a variable X_i and has two labeled outgoing edges: a 1-edge and a 0-edge. The labeling of the OBDD nodes respects a global ordering of the variables $\mathbf{X}$: if there is an edge from a node labeled X_i to a node labeled X_j, then X_i must come before X_j in the ordering. To evaluate the OBDD on an instance $\mathbf{x}$, start at the root node of the OBDD and let x_i be the value of variable X_i that labels the current node. Repeatedly follow the x_i-edge of the current node, until a sink node is reached. Reaching the 1-sink means $\mathbf{x}$ is evaluated to 1 and reaching the 0-sink means $\mathbf{x}$ is evaluated to 0 by the OBDD.

and Theobald 1998; Wegener 2000). Circuits, as in Figure 2b, are not in general tractable as OBDDs, although we will later seek to obtain tractable circuits through knowledge compilation, a subject which we will revisit in more depth in Section 4. Note further that OBDDs are also circuits that are notated more compactly.[4]

Our first goal is to obtain a tractable circuit representation of a given neuron. First, consider the following class of threshold-based linear classifiers.

Definition 1. *Let $\mathbf{X}$ be a set of binary features where each feature X in $\mathbf{X}$ has a value $x \in \{0, 1\}$. Let $\mathbf{x}$ denote an instantiation of variables $\mathbf{X}$. Consider functions f that map instantiations $\mathbf{x}$ to a value in $\{0, 1\}$. We call f a linear classifier if it has the following form:*

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{x \in \mathbf{x}} w_x \cdot x \geq T \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where T is a threshold, $x \in \mathbf{x}$ is the value of variable X in instantiation $\mathbf{x}$, and where w_x is the real-valued weight associated with value x of variable X.

Note that such classifiers are also Boolean functions. The following result, due to (Chan and Darwiche 2003), gives us a way of obtaining a tractable circuit representing the Boolean function of such classifiers.

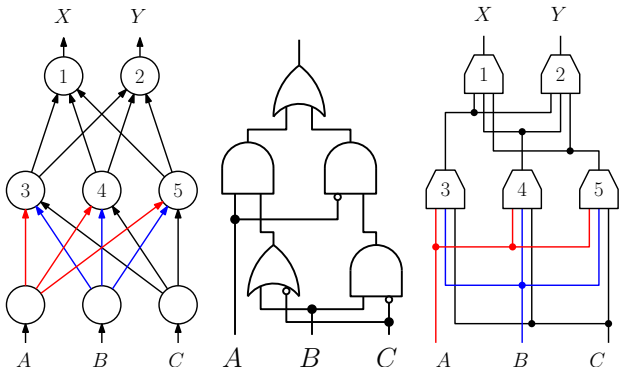
Theorem 1. *A linear classifier in the form of Equation 2 can be represented by an OBDD of size $O(2^{\frac{n}{2}})$ nodes, which can be computed in $O(n2^{\frac{n}{2}})$ time.*

(Chan and Darwiche 2003) further provided an algorithm to obtain the result of Theorem 1, although much more efficiently than what the bounds suggest. It was originally for compiling naive Bayes classifiers to Ordered Decision Diagrams (ODDs). However, this algorithm applies to any classifier of the form given by Equation 2, which includes naive Bayes classifiers, but also logistic regression classifiers, as well as neurons with step activation functions (Elkan 1997).

Compiling a linear classifier such as a neuron or a naive Bayes classifier is NP-hard (Shih, Choi, and Darwiche 2018b), hence algorithms, such as the one from (Chan and Darwiche 2003), are unlikely to have much tighter bounds. However, we can significantly tighten this bound if we make additional assumptions about the classifier's parameters.

Theorem 2. *Consider a linear classifier in the form of Equation 2, where the weights w_x and threshold T are integers. Such a classifier can be represented by an OBDD of size $O(nW)$ nodes, and compiled in $O(nW)$ time, where $W = |T| + \sum_x |w_x|$ is a sum of absolute values.*

While this result is known, Appendix A provides a construction for completeness.[5] Note that the integrality assumption of Theorem 2 can be applied to classifiers with real-valued weights by multiplying the parameters by a constant and then truncating (i.e., the parameters have fixed precision). As we show later, this pseudo-polynomial time algorithm enables the compilation of neurons, and ultimately neural networks, with *hundreds* of features. This is in contrast to the preliminary work of (Choi et al. 2019), which was based on the algorithm of (Chan and Darwiche 2003) for compiling neurons that scaled only to *dozens* of features.



(a) neural network (b) circuit of neuron (c) circuit of network

Figure 3: A neural network, the circuit of a single neuron, and the circuit of the original network. Wires highlighted in red and blue correspond to the inputs A and B, respectively.

Now that we can compile each neuron into a (tractable) Boolean circuit, the whole neural network will then induce a Boolean circuit as illustrated in Figure 3. That is, for the given neural network in Figure 3a, each neuron is compiled into a Boolean circuit as in Figure 3b. The circuits for neurons are then connected according to the neural network structure, leading to the Boolean circuit in Figure 3c, where the circuit of each neuron is portrayed as a block.

Using the algorithm of (Chan and Darwiche 2003), the Boolean circuit that we obtain from a neuron is tractable. The network's Boolean circuit, that we construct from the Boolean circuits of the neurons, may not be tractable however. To use the explanation and verification techniques proposed in (Shih, Choi, and Darwiche 2018b; Shih, Choi, and Darwiche 2018a), we require a tractable circuit; cf. (Darwiche and Hirth 2020). We next show how to obtain such a circuit using tools from the field of knowledge compilation.

4 Tractability via Knowledge Compilation

In this section, we provide a short introduction to the domain of knowledge compilation, and then show how we can compile a neural network into a tractable Boolean circuit.

We follow (Darwiche and Marquis 2002), which considers tractable representations of Boolean circuits, and the trade-offs between succinctness and tractability. In particular, they consider Boolean circuits of and-gates, or-gates and inverters, but where inverters only appear at the inputs (hence the inputs of the circuit are variables or their negations). This sub-class of circuits is called *Negation Normal Form* (NNF) circuits. Any circuit with and-gates, or-gates and inverters can be efficiently converted into an NNF cir-

[4] An OBDD node labeled by variable X and with children f_x and $f_{\bar{x}}$ is equivalent to the circuit fragment $(x \wedge f_x) \vee (\bar{x} \wedge f_{\bar{x}})$.

[5] This result appears, for example, as an exercise in https://www.cs.ox.ac.uk/people/james.worrell/lectures.html. This result also falls as a special case of (Chubarian and Turan 2020), which showed how to compile tree-augmented naive Bayes classifiers into OBDDs, where a naive Bayes classifier is a special case.

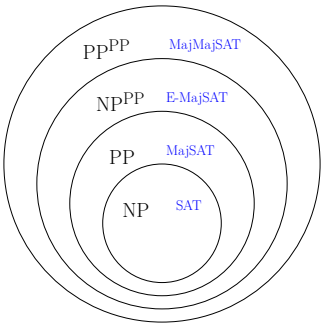


Figure 4: Containment of four complexity classes: $\mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{NP}^{\mathrm{PP}} \subseteq \mathrm{PP}^{\mathrm{PP}}$. Their canonical problems are labeled in blue.

cuit while at most doubling its size.

By imposing properties on the structure of NNF circuits, one can obtain greater tractability (the ability to perform certain operations in polytime) at the possible expense of succinctness (the size of the resulting circuit). To motivate this trade-off, consider Figure 4, which highlights the containment relationship between four complexity classes. The "easiest" class is NP, and the "hardest" class is $\mathrm{PP}^{\mathrm{PP}}$. The canonical problems that are complete for each class all correspond to queries on Boolean expressions. One popular computational paradigm for solving problems in these classes is to reduce them to the canonical problem for that class, and to compile the resulting Boolean expressions to circuits with the appropriate properties.[6][7] For example, (Oztok, Choi, and Darwiche 2016) shows how to solve $\mathrm{PP}^{\mathrm{PP}}$-complete problems by reduction to MajMajSAT queries on a specific tractable class of Boolean circuits.

Consider now a property on NNF circuits called *decomposability* (Darwiche 2001a). This property asserts that the sub-circuits feeding into an and-gate cannot share variables. An NNF circuit that is decomposable is said to be in Decomposable Negation Normal Form (DNNF). In a DNNF circuit, testing whether the circuit is satisfiable can be done in time *linear* in the size of the circuit. Another such property is *determinism* (Darwiche 2001b). This property asserts that for each or-gate, if the or-gate outputs 1 then exactly one of its input is 1. A DNNF circuit that is also deterministic is called a d-DNNF. The circuit in Figure 2b is an example of a d-DNNF circuit. In a d-DNNF circuit, counting the number of assignments that satisfy the circuit can be done in time *linear* in the size of the circuit, assuming the circuit also satisfies smoothness (Darwiche 2003).[8] Hence, with these first two properties, we can solve the canonical problems in the two "easiest" classes illustrated in Figure 4.

A more recently proposed class of circuits is the Sentential Decision Diagram (SDD) (Darwiche 2011; Xue, Choi, and Darwiche 2012; Choi and Darwiche 2013). SDDs are a subclass of d-DNNF circuits that assert a stronger form of decomposability, and a stronger form of determinism. SDDs subsume OBDDs and are exponentially more succinct (Bova 2016). SDDs support polytime conjunction and disjunction. That is, given two SDDs α and β, there is a polytime algorithm to construct another SDD γ that represents $\alpha \wedge \beta$ or $\alpha \vee \beta$.[9] Further, SDDs can be negated in linear time.[10]

These polytime operations allow a simple algorithm for compiling a Boolean circuit with and-gates, or-gates and inverters into an SDD. We first obtain an SDD for each circuit input. We then traverse the circuit bottom-up, compiling the output of each visited gate into an SDD by applying the corresponding operation to the SDDs of the gate's inputs.

SAT and MajSAT can be solved in linear time on SDDs. Further properties on SDDs allow the problems E-MajSAT and MajMajSAT, the two hardest problems illustrated in Figure 4, to be also solved in time linear in the size of the SDD (Oztok, Choi, and Darwiche 2016). In our experiments, we compiled the Boolean circuits of neural networks into standard SDDs as this was sufficient for efficiently supporting the explanation and verification queries we are interested in.

5 On the Robustness of Classifiers

Neural networks are now ubiquitous in machine learning and artificial intelligence, but it is increasingly apparent that neural networks learned in practice can be fragile. That is, they can be susceptible to misclassifying an instance after small perturbations have been applied to it (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2014; Moosavi-Dezfooli, Fawzi, and Frossard 2016; Wang et al. 2018; Zhang, Zhang, and Hsieh 2019). Next, we show how compiling a neural network into a tractable circuit can provide one with the ability to analyze the robustness of a neural network's decisions.

We consider first the robustness of a binary classifier's decision to label a given instance 0 or 1. The notion of robustness that we consider is based on the following question: how many features do we need to flip from 0 to 1 or 1 to 0, before the classifier's decision flips? That is, we consider the robustness of a given instance to be the (Hamming) distance to the closest instance of the opposite label.

Definition 2 (Instance-Based Robustness)**.** *Consider a Boolean classification function $f : \{0,1\}^n \rightarrow \{0,1\}$ and a given instance $\mathbf{x}$. The robustness of the classification of $\mathbf{x}$ by f, denoted by $\mathsf{r}_f(\mathbf{x})$, is defined as follows. If f is a trivial function (true or false), then $\mathsf{r}_f(\mathbf{x}) = \infty$. Otherwise,*

$$\mathsf{r}_f(\mathbf{x}) = \min_{\mathbf{x}' : f(\mathbf{x}') \neq f(\mathbf{x})} d(\mathbf{x}, \mathbf{x}')$$

where $d(\mathbf{x}', \mathbf{x})$ denotes the Hamming distance between $\mathbf{x}'$ and $\mathbf{x}$, i.e., the number of variables on which $\mathbf{x}$ and $\mathbf{x}'$ differ.

This notion of robustness was also considered by (Shih, Choi, and Darwiche 2018a), who also assumed binary (or

[6] For more on this paradigm, see http://beyondnp.org.

[7] For a video tutorial on this paradigm, "On the role of logic in probabilistic inference and machine learning," see https://www.youtube.com/watch?v=xRxP2Wj4kuA

[8] Counting how many assignments satisfy a given circuit allows us to tell whether a majority of them satisfy the circuit (MajSAT).

[9] If s and t are the sizes of input SDDs, then conjoining or disjoining the SDDs takes $O(s \cdot t)$ time, although the resulting SDD may not be compressed (Van den Broeck and Darwiche 2015).

[10] In our case study in Section 6, we used the open-source SDD package available at http://reasoning.cs.ucla.edu/sdd/.

categorical) features. Other notions of robustness are discussed in (Leofante et al. 2018), for real-valued features.

Given a classification function f, we refer to an instance $\mathbf{x}$ as being k-robust if $\mathsf{r}_f(\mathbf{x}) = k$, i.e., it takes at least k flips of the features to flip the classification. In general, it is intractable to compute the robustness of a classification, unless P=NP. Consider the following decision problem:

D-ROBUST: Given function f, instance $\mathbf{x}$, and integer k, is $\mathsf{r}_f(\mathbf{x}) \geq k$?

Theorem 3. D-ROBUST *is coNP-complete.*

Proof. Let f be a DNF formula over n variables, and let $\mathbf{x}$ be an arbitrary instantiation of these variables. Further, let ℓ be a literal of a new variable. The problem is in coNP, because it is polytime falsifiable: given a counterexample $\mathbf{x}'$, we can check in polytime that $D(\mathbf{x}, \mathbf{x}') < k$ and that $f(\mathbf{x}') \neq f(\mathbf{x})$. The problem is coNP-hard since f is a tautology iff the DNF $f \vee \ell$ and instantiation $\mathbf{x}$ has robustness at least $n + 1$ (i.e., infinite). See also Footnote 14. □

Theorem 3 implies that we can use a SAT solver to compute the robustness of an instance relative to a Boolean function.[11] Given a tractable circuit (in particular, an OBDD), this question can also be answered in time *linear* in the size of the circuit (Shih, Choi, and Darwiche 2018a).[12] We employ the algorithm given by (Shih, Choi, and Darwiche 2018a) in our case studies in Section 6.

Next, rather than consider the robustness of just one classification, we can consider the *average* robustness of a classification function, over all possible inputs. In other words, we consider the *expected* robustness of a classifier, under a uniform distribution of its inputs.

Definition 3 (Model-based Robustness)**.** *Consider a Boolean classification function $f : \{0,1\}^n \rightarrow \{0,1\}$. The model robustness of f is defined as:*

$$\mathsf{mr}(f) = \frac{1}{2^n} \sum_{\mathbf{x}} \mathsf{r}_f(\mathbf{x})$$

Let f be the classification function whose robustness we want to assess. If $f(\mathbf{x}) = 1$ we refer to $\mathbf{x}$ as a positive instance; otherwise $f(\mathbf{x}) = 0$ and we refer to $\mathbf{x}$ as a negative instance. We propose Algorithm 1 for computing the model robustness of a classifier over all *positive* instances (the robustness of negative instances can be computed by invoking Algorithm 1 on function $\neg f$).[13]

Algorithm 1 `model-robustness`(f)

input: A classifier's Boolean function f
output: The (positive) model-based robustness $\mathsf{mr}(f)$
main:

$M, h_1 \leftarrow 0, f$
for k from 2 to n **do**
  $h_k \leftarrow \bigwedge_X (h_{k-1}|x \wedge h_{k-1}|\bar{x})$
  $f_{k-1} \leftarrow h_{k-1} \wedge \neg h_k$
  $M \leftarrow M + (k-1) \cdot \mathsf{model\_count}(f_{k-1})$
end for
$M \leftarrow M + n \cdot \mathsf{model\_count}(h_n)$ {since $f_n \equiv h_n$}
return M

Our algorithm is based on computing the set of functions h_k, which are the Boolean functions representing all positive instances $\mathbf{x}$ that have robustness k or higher. That is, h_k represents all instances $\mathbf{x}$ where $f(\mathbf{x}) = 1$ and where $\mathsf{r}_f(\mathbf{x}) \geq k$. First, $h_1 = f$. For $k = 2$ we have: $h_2 = \bigwedge_X (f|x \wedge f|\bar{x})$, where $f|x$ denotes the conditioning of f on value x, i.e., the function that we would obtain by setting X to true (replace every occurrence of X with true, and in the case of $f|\bar{x}$, replace X with false). Say that $\mathbf{x}$ is an instance of $f|x \wedge f|\bar{x}$, and thus $f(\mathbf{x}) = 1$ and $f(\mathbf{x})$ remains 1 no matter how we set X. By taking the conjunction across all variables X, we obtain all instances $\mathbf{x}$ whose output would not flip after flipping any single feature X. Next, consider the robustness of the instances of h_2. Some of these instances $\mathbf{x}$ will become 1-robust with respect to h_2. These instances are in turn 2-robust with respect to the original function f. More generally, we can compute h_k from h_{k-1}, via $h_k = \bigwedge_X (h_{k-1}|x \wedge h_{k-1}|\bar{x})$. We can now compute the functions f_k representing all of the k-robust examples of f, via $f_k = h_k \wedge \neg h_{k+1}$. The model count of f, denoted by $\mathsf{model\_count}(f)$, is the number of instances $\mathbf{x}$ satisfying a Boolean function f. We can then compute the model robustness of f by:

$$\mathsf{mr}(f) = \frac{1}{2^n} \sum_{k=1}^{n} \mathsf{model\_count}(f_k) \cdot k.$$

Consider now the *most* robust instances of a function f.

Definition 4 (Maximum Robustness)**.** *Consider a Boolean classification function $f : \{0,1\}^n \rightarrow \{0,1\}$, where f is non-trivial. The maximum robustness of f is defined as:*

$$\mathsf{maxr}(f) = \max_{\mathbf{x}} \mathsf{r}_f(\mathbf{x}).$$

Note that the instances $\mathbf{x}$ of h_k is a subset of the instances of h_{k-1}, as computed in Algorithm 1. Hence, the model

[11] For a given k, we can determine if $r_f(x) \leq k$ by first encoding the set of instances within a distance of k away from x as a CNF formula ϕ, using a standard encoding. We then encode the value of $f(x)$ and the neural network's classification function as a CNF formula Δ, using a technique similar to that by (Narodytska et al. 2018). The formula $\phi \wedge \Delta$ is then satisfiable iff $r_f(x) \leq k$. Finally, we iterate over all possible values of k (or perform binary search), so the instance-based robustness is just the smallest value of k such that $r_f(x) \leq k$ (i.e., $\phi \wedge \Delta$ is SAT).

[12] The robustness of an instance $y, \mathbf{x}$ can be computed by the following recurrence, which recurses on the structure of an OBDD: $\mathsf{r}_f(y, \mathbf{x}) = \min\{\mathsf{r}_{f|y}(\mathbf{x}), 1 + \mathsf{r}_{f|\bar{y}}(\mathbf{x})\}$ where $\mathsf{r}_f(\mathbf{x}) = 0$ if f is false and $\mathsf{r}_f(\mathbf{x}) = \infty$ if f is true.

[13] Recently, (Baluta et al. 2019) proposed an approach for estimating robustness using approximate model-counting, with PAC-style guarantees. Their approach scaled to 10×10 digits datasets; our exact approach scales to 16×16 digits datasets in Section 6.

count of h_k will decrease as we increase k. At a large enough k, then h_{k+1}, and hence also f_{k+1}, will have no models and will equal false. At this point, we know k is the maximum robustness, and we can stop Algorithm 1 early. Further, this f_k gives us the set of examples that are most robust (requiring the most number of features to flip).[14]

Finally, we observe that model-based robustness appears to be computationally more difficult than instance-based robustness. In particular, the model-robustness over positive instances can be shown to be a PP-hard problem.

D-P-MODEL-ROBUST: Given function f and integer k, is the positive model robustness of f at least k?

Theorem 4. D-P-MODEL-ROBUST *is PP-hard.*

Proof. Let f be a CNF formula over n variables and let ℓ be a literal of a new variable. Note that model counting is #P-complete for CNF (Valiant 1979), and that any positive instance of the CNF $f \wedge \ell$ has a robustness of 1. It follows that $\mathsf{model\_count}(f) \geq k$ iff the positive model robustness of $f \wedge \ell$ is at least $k/2^{n+1}$. □

To compute model-based robustness using Algorithm 1, we must be able to negate, conjoin and condition on Boolean functions, as well as compute their model count. Given a circuit represented as an SDD, operations such as negation and counting the models of an SDD can be done in time *linear* in the size of the SDD. Conjoining two SDDs of size s and t takes time $O(st)$ although a sequence of conjoin operations may still take exponential time, as in Algorithm 1.

6 A Case Study

We next provide a case study in explaining and verifying a convolutional neural network via knowledge compilation.

6.1 (Binary) Convolutional Neural Networks

In our case study, we consider *binary* convolutional neural networks (binary CNNs).[15] That is, if we assume binary inputs and step activations, then the outputs of all neurons are binary, and the output of the network itself is also binary. Hence a binary CNN represents a Boolean function. We can construct a Boolean circuit representing a binary CNN, and then compile it to a tractable one as described in Section 4.

Our binary CNNs contain three types of layers:

- *convolution + step layers:* a convolution layer consists of a set of filters, that can be used to detect local patterns in an input image. Typically, a ReLU unit is applied to the output of a filter. In a binary CNN, we assume step activations (whose parameters are trained first using sigmoid activations, then replacing them with step activations);
- *max-pooling layers:* a max-pooling layer can be used to reduce the dimension of an image, helping to reduce the overall computational and statistical demands. In a binary CNN, if the inputs of a max-pooling layer is 0 or 1, then the "max" reduces to a logical "or";
- *fully-connected layers:* if the inputs are binary and if we use step activations, then each neuron represents a Boolean function, as in Section 3.

6.2 Experimental Setup

We consider the USPS digits dataset of handwritten digits, consisting of 16×16 pixel images, which we binarized to black and white (Hull 1994). Here, we performed binary classification using different pairs of digits. We first trained a CNN using sigmoid activations, using TensorFlow. We replaced the sigmoid activations with step activations, to obtain a binary CNN that we compiled into a tractable circuit. In particular, we compiled the binary CNN into a Sentential Decision Diagrams (SDD). We shall subsequently provide analyses of the binary CNN, via queries on the SDD.

More specifically, we created two convolution layers, each with stride size 2. We first swept a 3×3 filter on the original 16×16 image (resulting in a 7×7 grid), followed by a second 2×2 filter (resulting in a 3×3 grid). These outputs were the inputs of a fully-connected layer with a single output. We did not use max-pooling as the dimension was reduced enough by the convolutions. Finally, we optimized a sigmoid cross-entropy loss using the Adam optimizer.

The SDD circuits compiled from neural networks in the following experiments are exact as the compilation process utilized the exact neuron compiler based on (Chan and Darwiche 2003).[16] We later also evaluate the newly proposed, approximate neuron compiler, which has a pseudo-polynomial time complexity, showing the trade-off it leads to between classification accuracy and size of compilation.

6.3 Explaining Decisions

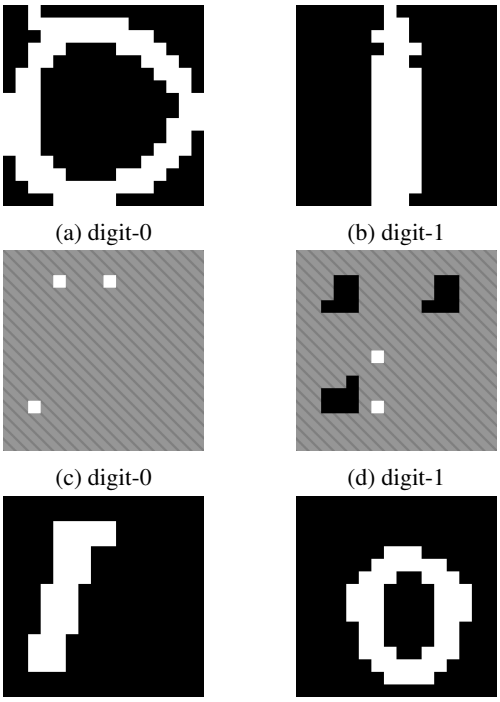
We consider how to explain *why* a neural network classified a given instance positively or negatively. In particular, we consider *prime-implicant explanations* (PI-explanations), as proposed by (Shih, Choi, and Darwiche 2018b); see also (Ignatiev, Narodytska, and Marques-Silva 2019a; Darwiche and Hirth 2020). Say that an input image $\mathbf{x}$ is classified positively, i.e., as a digit-1. A PI-explanation returns the smallest subset $\mathbf{y}$ of the inputs in $\mathbf{x}$ that render the remaining inputs irrelevant. That is, once you fix the pixel values $\mathbf{y}$, the values of the other pixels do not matter—the network will always classify the instance as a digit-1.[17]

[14] Note that if f is a non-trivial Boolean function, then f_n must be false. Suppose f_n were not false, and that $\mathbf{x}$ is an instance of f_n. This means we can flip any and all variables of $\mathbf{x}$, and it would always be an example of f. This implies that f must have been true, and hence a trivial function.

[15] A number of binary variations of neural networks have been proposed in the literature. The XNOR-Networks of (Rastegari et al. 2016) are another binary variation of CNNs, which also assumes binary weights. The binarized neural networks (BNNs) of (Hubara et al. 2016) have binarized parameters and activations. In work closely related to ours, (Narodytska et al. 2018) studied the verification of BNNs, using SAT solvers, as discussed in Section 1.

[16] An updated version of the compiler used in these experiments is available at https://github.com/art-ai/nnf2sdd. The compiler integrates the newly proposed neuron compiler with pseudo-polynomial time complexity.

[17] Consider in contrast "Anchors," recently proposed by (Ribeiro, Singh, and Guestrin 2016a; Ribeiro, Singh, and Guestrin 2018). An



(a) digit-0 (b) digit-1

(c) digit-0 (d) digit-1

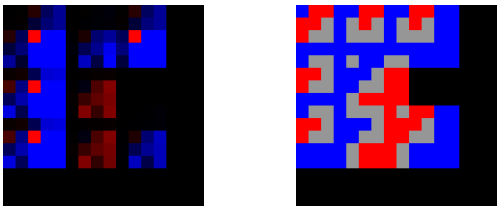
(e) a 1 labeled as a 0 (f) a 0 labeled as a 1

Figure 5: A correctly classified digit-0 and digit-1 from the test set (first row), the corresponding PI-explanations (second row, gray striped regions represent "don't care's"), and the corresponding fooling images (third row).

We first trained a CNN to distinguish between digit-0 and digit-1 images, which achieved 98.74% accuracy. The resulting SDD had 5,900 nodes and 28,735 edges. We took one correctly classified instance of each digit from the test set, shown in Figures 5a & 5b. The shortest PI-explanations for these two images are displayed in Figures 5c & 5d. In Figure 5c, the PI-explanation consists of three white pixels. Once we fix these three pixels, the network will always classify the image as a digit-0, no matter how the pixels in the gray region are set. Similarly, Figure 5d sets three black patches of pixels to the left and right, and sets two center pixels to white, which is sufficient for the network to always classify the image as a digit-1.

These PI-explanations provide strong guarantees: the pixels in the gray region can be manipulated in *any* way and the classification would still not change. They are so strong in fact that one can easily create counterexamples to fool the network. In Figures 5e & 5f, we fill in the remaining pixels in such a way that the digit-0 image looks like a digit-1, and vice versa. The network classifies these new images incorrectly because it is misled by the subset of pixels shown in the PI-explanation of Figures 5c & 5d. Using this method, we can readily generate counterexamples such as these. We obtained similar results with other pairs of digits.

anchor for an instance $\mathbf{x}$ is a subset of the instance that is highly likely to be classified with the same label, no matter how the missing features are filled in (according to some distribution). In contrast, PI-explanations are *exact*.



(a) Marginal grid (b) Unateness grid

Figure 6: Visualizations of the relationship between the output of the network and each individual pixel.

6.4 Explaining Model Behavior

To explain the network's behavior as a whole (and not just per instance), we provide two visualizations of how each pixel contributes to the classification decision: a *marginal grid* and a *unateness grid*. Figure 6a is a *marginal grid*, which highlights the marginals of the output neuron, i.e., the probability that each pixel is white given that the output of the network is digit-1. In general, it is intractable to compute such marginals, which naively entails enumerating all 2^{256} possible input images and then checking the network output. If we can compile a neural network's Boolean function into a tractable circuit, like an SDD, then we can compute such marginals in time linear in the size of the circuit.

In Figure 6a, red pixels correspond to marginals greater than $\frac{1}{2}$, and redder pixels are closer to one. Blue pixels correspond to marginals less than $\frac{1}{2}$, and bluer pixels are closer to zero. The grid intensities have been re-scaled for clarity. Not surprisingly, we find that if the output of the network is high (indicative of a digit-1), then it is somewhat more likely that the pixels in the middle are set to white.

Figure 6b is a *unateness grid*, which identifies pixels that sway the classification in one direction only. Red pixels are positively unate (monotone), so turning them from off to on can only flip the classification from digit-0 to digit-1. Blue pixels are negatively unate, i.e., turning them from off to on can only flip the classification from digit-1 to digit-0. Black pixels are ignored by the network completely. Finally, gray pixels do not satisfy any unateness property. In general, determining whether an input of a Boolean function is unate/monotone or unused are computationally hard problems. In tractable circuits such as SDDs, they are queries that can be performed in time polynomial in the circuit size.

In Figure 6b, the majority of pixels are unate (monotone), suggesting that the overall network behavior is still relatively simple. Note that there are many unused pixels on the right and bottom borders. This can be explained by the lack of padding (i.e., given the filter size and stride length, no filter takes any of these pixels as inputs). There is another block of unused pixels closer to the middle. On closer inspection, these pixels are unique to one particular filter in the second convolution layer (no other filter depends on their values). In the tractable circuit of the output neuron, we find that the circuit does not essentially depend on the output of this filter. Thus, the output of the network does not depend on the

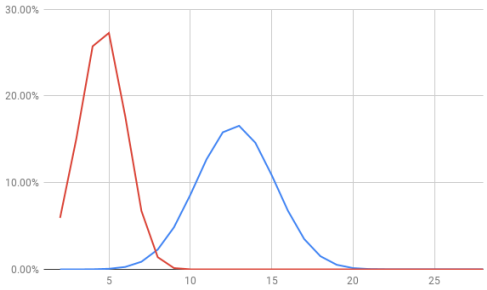


Figure 7: Level of robustness k vs. proportion of instances. Net 1 is plotted in blue (right) and Net 2 in red (left).

values of any of these pixels. Note that deciding whether an input of a neuron is unused is an NP-hard problem.[18] However, given a tractable circuit such as an SDD, this question can be answered in time linear in the size of the circuit.

We emphasize a few points now. First, this (visual) analysis is enabled by the tractability of the circuit, which allows marginals to be computed and unate pixels to be identified efficiently. Second, the analysis also emphasizes that the network is not learning the conceptual differences between a digit-0 and a digit-1. It is identifying subsets of the pixels that best differentiate between images of digit-0 and digit-1 from the training set with high accuracy. This perhaps explains why it is sometimes easy to "fool" neural networks, which we demonstrated in Figure 5.

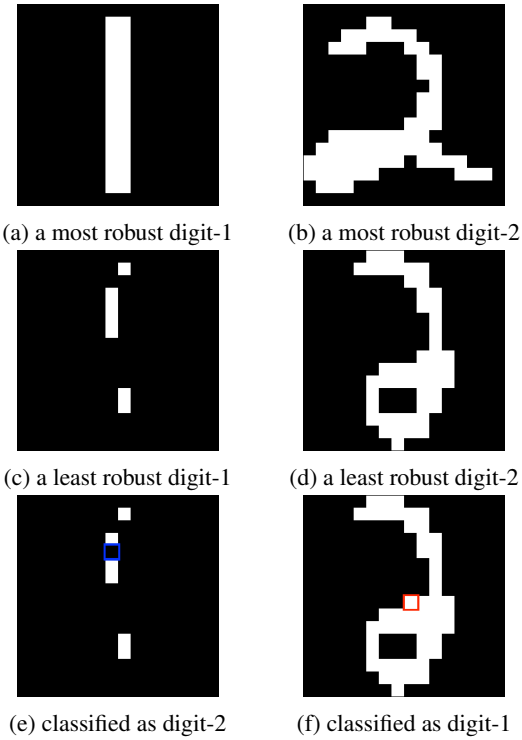
6.5 Analyzing Classifier Robustness

Next, we provide a case study in analyzing CNNs based on their robustness. We consider the classification task of discriminating between a digit-1 and a digit-2. First, we trained two CNNs with the same architectures (as described earlier), but using two different parameter seeds. We achieved 98.18% (Net 1) and 96.93% (Net 2) testing accuracies. The SDD of Net 1 had 1,298 nodes and a size of 3,653. The SDD of Net 2 had 203 nodes and a size of 440.[19] Net 1 obtained a model-robustness of 11.77 but Net 2 only obtained a robustness of 3.62. For Net 2, this means that on average, 3.62 pixel flips are needed to flip a digit-1 classification to digit-2, or vice versa. Moreover, the maximum-robustness of the Net 1 was 27, while that of Net 2 was only 13. For Net 1, this means that there is an instance that would not flip unless you flipped (the right) 27 pixels. These are two networks which are similar in terms of accuracy (differing by only 1.25%), but *very* different when compared by robustness.

Figure 7 further highlights the differences between these two networks by the level of robustness k. On the x-axis, we increase the level of robustness k (up to the max of 27), and on the y-axis we measure the proportion of instances with robustness k, i.e., we plot $2^{-256} \cdot \mathsf{model\_count}(f_k)$, as in Section 5. Clearly, the first network more robustly classifies

[18]This reduction is similar to the one showing that compiling a linear classifier is NP-hard (Shih, Choi, and Darwiche 2018b).

[19]The size of a decision node in an SDD is the number of its children. The size of an SDD is the aggregate size of its nodes.



(a) a most robust digit-1 (b) a most robust digit-2

(c) a least robust digit-1 (d) a least robust digit-2

(e) classified as digit-2 (f) classified as digit-1

Figure 8: Visualizations of robustness.

a larger number of instances. Given two networks with similar accuracies, we prefer the one that is more robust, as it would be more resilient to adversarial perturbations and to noise. When we compute the average instance-based robustness of testing instances, Net 1 obtains an average of 4.47, whereas Net 2 obtains a lower average of 2.61, as expected.

Next, we consider in more depth Net 2, which again had a test set accuracy of 96.93%. First, we visualize the most robust and the least robust instances of the CNN. Figures 8a & 8b depict an example of a most robust digit-1 and digit-2, from the testing set. Similarly Figures 8c & 8d depict an example of a least robust digit-1 and digit-2, both having robustness 1. For these latter two instances, it suffices to flip a single pixel in each image, for the classifier to switch its label. These perturbations are given in Figures 8e & 8f. Finding training examples that have low-robustness can help finding problematic or anomalous instances in the dataset, or otherwise indicate weaknesses of the learned classifier. Finding training examples that have high-robustness provides an insight into which instances that the classifier considers to be prototypical of the class.

6.6 Pseudo-Polynomial Neuron Compilation

Finally, we evaluate the pseudo-polynomial time algorithm of Theorem 2, for compiling a neuron into an SDD (and more specifically into an OBDD). This algorithm runs in polynomial time when the precision of the neuron's parameters are fixed. We consider the same USPS digits dataset used in our case study. Instead of training a neural network, we train a single neuron with $16 \times 16 = 256$ inputs to clas-

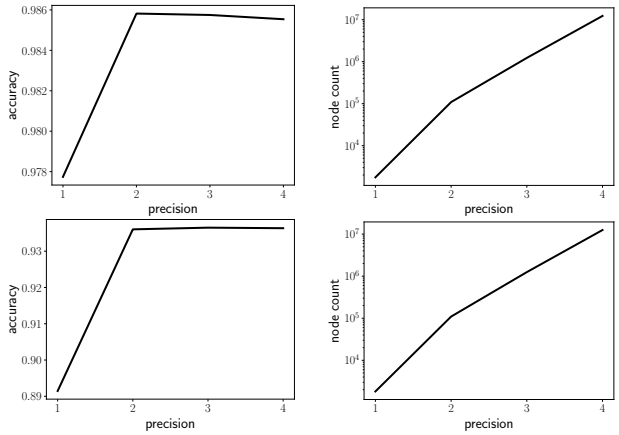


Figure 9: Precision (in digits) versus test-set accuracy and node count. Upper row is an average over 10 random parameter seeds for 0-vs-1. Lower row is an overage over all 45 pairs of digits.

sify a digit, which corresponds (roughly) to logistic regression. Having $n = 256$ inputs is well beyond the scope of the exact algorithm proposed by (Chan and Darwiche 2003), whose worst-case running time can be $O(2^{\frac{n}{2}})$.

Consider first Figure 9 (upper row). Here, we trained a single neuron as a 0-versus-1 classifier, and averaged over 10 different parameter seeds. On the x-axis, we increase the number of digits of precision in the weights, and on the y-axes we measure test-set accuracy and the size of the OBDD in terms of node count. First, we find that with 2 digits of precision, we maintain a high 98.6% accuracy. Next, we observe that even with a single digit of precision, we can still maintain around a 97.8% accuracy. Next, we find that as we increase the digits of precision, the size of the resulting OBDD grows exponentially, as expected since compiling a neuron to an OBDD is NP-hard (Shih, Choi, and Darwiche 2018b). All 10 cases failed (out-of-memory) for 5 digits of precision. These observations suggest that a high degree of precision is not necessary to obtain good predictive performance, as observed by (Rastegari et al. 2016; Hubara et al. 2016) as well, in the more extreme case of binarized weights. We see a similar story in Figure 9 (lower row), where we have averaged over all 45 pairs of digits.

7 Conclusion and Discussion

We proposed a knowledge compilation approach for explaining and verifying the behavior of a neural network. We considered in particular neural networks with 0/1 inputs and step activation functions. Such networks have neurons that correspond to Boolean functions. The network itself also corresponds to a Boolean function, which maps an input feature vector into a class. We showed how to compile the Boolean function of each neuron and the network itself into a tractable circuit in the form of a Sentential Decision Diagram (SDD). We also introduced a pseudo-polynomial time algorithm that compiles neurons into tractable circuits, which can scale to neurons with hundreds of inputs when fixing the precision of neuron weights. We also developed new queries and algorithms for analyzing the robustness of a Boolean function. In a case study, we explained and analyzed the robustness of binary CNNs for classifying handwritten digits, and empirically evaluated the pseudo-polynomial time algorithm for compiling neurons.

As highlighted in Section 4, the field of knowledge compilation studies the trade-offs between succinctness (the size of a circuit) and tractability (the number of polytime queries supported by the circuit). In the context of explainable AI, this amounts to a trade-off between scalability and amenability to analysis. In terms of computational complexity, queries such as instance-based robustness correspond to the complexity class NP and can therefore be tackled using SAT/SMT/MILP approaches. Queries such as model-based robustness correspond to the complexity class PP and require approaches based on model counting and knowledge compilation (such as the approach we proposed in this paper). Since $\mathrm{NP} \subseteq \mathrm{PP}$, approaches based on SAT solving (NP) are inherently more scalable, e.g., (Katz et al. 2017; Narodytska et al. 2018; Ignatiev, Narodytska, and Marques-Silva 2019a), whereas approaches based on model counting and knowledge compilation offer more powerful types of analyses (Shih, Choi, and Darwiche 2018b; Baluta et al. 2019; Audemard, Koriche, and Marquis 2020; Darwiche and Hirth 2020). Another class of promising approaches are the ones based on approximate model counting (Baluta et al. 2019), which represent an interesting compromise between scalability and analysis.

A Proof of Theorem 2

Consider a neuron with inputs $X_1, \ldots, X_n$. Setting the inputs $X_1, \ldots, X_i$ results in a smaller sub-classifier (or sub-neuron) over inputs $X_{i+1}, \ldots, X_n$. No matter how we set the inputs $X_1, \ldots, X_i$ the resulting sub-classifier is identical (has the same weights) except for the threshold being used. Two different settings of inputs $X_1, \ldots, X_i$ may lead to identical sub-classifiers with the same threshold. Say we set variables from X_1 to X_n. There are at most $2W$ possible valid thresholds, so there are at most $2nW$ possible sub-classifiers that we can see while setting variables.

Consider an $n \times W$ matrix A where cell $A[i][j]$ is associated with the sub-classifier where variable X_i is about to be set, and where j is the threshold being used. If X_i is set to 1, we obtain the sub-classifier at $A[i+1][j+w_i]$ where w_i is the (integer) weight of feature X_i. If X_i is set to 0, we obtain the sub-classifier at $A[i+1][j]$. Each cell $A[i][j]$ thus represents an OBDD node for the corresponding sub-problem, whose hi- and lo-children are known. We add an $(n+1)$-th layer where every sub-classifier with threshold below 0 is $\bot$, and where every sub-classifier at or above 0 is $\top$. The root of the original neuron's OBDD is then found at $A[1][T]$, which we can extract and reduce if needed.

The size of matrix A bounds the size of the OBDD to $O(nW)$ nodes. Further, it takes constant time to populate each entry, and hence $O(nW)$ time to construct the OBDD.

Finally, we note that the above construction also follows the proof of Theorem 1 given by (Chan and Darwiche 2003), based also on identifying equivalence classes of sub-classifiers. Thus, we can view Theorem 2 as a tightening of the bounds of Theorem 1 for a special case (integer weights).

Acknowledgements

This work has been partially supported by NSF grant #ISS-1910317, ONR grant #N00014-18-1-2561, and DARPA XAI grant #N66001-17-2-4032.

References

Audemard, G.; Koriche, F.; and Marquis, P. 2020. On tractable XAI queries based on compiled representations. In *KR*.

Baehrens, D.; Schroeter, T.; Harmeling, S.; Kawanabe, M.; Hansen, K.; and Müller, K. 2010. How to explain individual classification decisions. *Journal of Machine Learning Research* 11:1803–1831.

Baluta, T.; Shen, S.; Shinde, S.; Meel, K. S.; and Saxena, P. 2019. Quantitative verification of neural networks and its security applications. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS)*, 1249–1264.

Bova, S. 2016. SDDs are exponentially more succinct than OBDDs. In *AAAI*, 929–935.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35:677–691.

Cadoli, M., and Donini, F. M. 1997. A survey on knowledge compilation. *AI Commun.* 10(3-4):137–150.

Chan, H., and Darwiche, A. 2003. Reasoning about Bayesian network classifiers. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 107–115.

Choi, A., and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *Proceedings of the 27th Conference on Artificial Intelligence (AAAI)*.

Choi, A.; Shi, W.; Shih, A.; and Darwiche, A. 2019. Compiling neural networks into tractable Boolean circuits. In *AAAI Spring Symposium on Verification of Neural Networks (VNN)*.

Chubarian, K., and Turan, G. 2020. Interpretability of bayesian network classifiers: Obdd approximation and polynomial threshold functions. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*.

Darwiche, A., and Hirth, A. 2020. On the reasons behind decisions. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *JAIR* 17:229–264.

Darwiche, A. 2001a. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

Darwiche, A. 2001b. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics* 11(1-2):11–34.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *J. ACM* 50(3):280–305.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of IJCAI*, 819–826.

Darwiche, A. 2014. Tractable knowledge representation formalisms. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press. 141–172.

Elkan, C. 1997. Boosting and naive Bayesian learning.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *CoRR* abs/1412.6572.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 4107–4115.

Hull, J. J. 1994. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence* 16(5):550–554.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019a. Abduction-based explanations for machine learning models. In *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI)*, 1511–1519.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019b. On relating explanations and adversarial examples. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 15857–15867.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification CAV*, 97–117.

Leofante, F.; Narodytska, N.; Pulina, L.; and Tacchella, A. 2018. Automated verification of neural networks: Advances, challenges and perspectives. *CoRR* abs/1805.09938.

Lipton, Z. C. 2018. The mythos of model interpretability. *Commun. ACM* 61(10):36–43.

Meinel, C., and Theobald, T. 1998. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer.

Moosavi-Dezfooli, S.; Fawzi, A.; and Frossard, P. 2016. DeepFool: A simple and accurate method to fool deep neural networks. In *Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2574–2582.

Narodytska, N.; Kasiviswanathan, S. P.; Ryzhyk, L.; Sagiv, M.; and Walsh, T. 2018. Verifying properties of binarized deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*.

Oztok, U.; Choi, A.; and Darwiche, A. 2016. Solving PP^{PP}-complete problems using knowledge compilation. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 94–103.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, 525–542.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016a. Nothing else matters: Model-agnostic explanations by identifying

prediction invariance. In *NIPS Workshop on Interpretable Machine Learning in Complex Systems*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016b. "why should i trust you?": Explaining the predictions of any classifier. In *Knowledge Discovery and Data Mining (KDD)*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2018. Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*.

Selman, B., and Kautz, H. A. 1996. Knowledge compilation and theory approximation. *J. ACM* 43(2):193–224.

Shih, A.; Choi, A.; and Darwiche, A. 2018a. Formal verification of Bayesian network classifiers. In *Proceedings of the 9th International Conference on Probabilistic Graphical Models (PGM)*.

Shih, A.; Choi, A.; and Darwiche, A. 2018b. A symbolic approach to explaining Bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*.

Shih, A.; Darwiche, A.; and Choi, A. 2019. Verifying binarized neural networks by angluin-style learning. In *SAT*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2013. Intriguing properties of neural networks. *CoRR* abs/1312.6199.

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8(3):410–421.

Van den Broeck, G., and Darwiche, A. 2015. On the role of canonicity in knowledge compilation. In *AAAI*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium*, 1599–1614.

Wegener, I. 2000. *Branching Programs and Binary Decision Diagrams*. SIAM.

Xue, Y.; Choi, A.; and Darwiche, A. 2012. Basing decisions on sentences in decision diagrams. In *AAAI*, 842–849.

Zhang, H.; Zhang, P.; and Hsieh, C. 2019. RecurJac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, 5757–5764.