Compiling Neural Networks into Tractable Boolean Circuits

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

Abstract

We show how to exploit tools and methods from the knowledge compilation literature in order to explain the behavior and verify the properties of neural networks. In particular, we show how to compile neural networks into tractable Boolean circuits to facilitate their efficient explanation and verification. We first show how to reduce a neural network over binary inputs and step activation functions into a Boolean circuit. We then compile this Boolean circuit into a tractable one (a core problem in the domain of knowledge compilation). Once we obtain such a tractable Boolean circuit, the explanation and verification of the original neural network can be done efficiently, as we illustrate through a case study.

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 2016; 2018; Lipton 2018).[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 may be required 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 in this paper a *knowledge compilation* approach for explaining and verifying the behaviors of neural network classifiers. Knowledge compilation is a sub-field of AI (Selman and Kautz 1996; Cadoli and Donini 1997; Darwiche and Marquis 2002; Darwiche 2014) that studies in part *tractable* Boolean circuits, and the trade-offs between succinctness and tractability. That is, 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 or 1) and that use step activation functions. Although such a network would be parameterized with real-valued weights, the network itself induces a purely Boolean function. We seek a *tractable Boolean circuit* that represents this Boolean function, which we propose to obtain in two steps.

For the first step, we observe that neurons with step activation functions have binary inputs and a binary output. Hence, each neuron induces its own Boolean function. Using the algorithm of (Chan and Darwiche 2003), we obtain a tractable circuit for a given neuron's Boolean function. Given the Boolean circuits of its neurons, a neural network then induces its own Boolean circuit, although not necessarily a tractable one. For the second step, we *compile* this circuit into a tractable one by enforcing additional properties on the circuit until operations of interest become tractable, as is commonly done in the domain of knowledge compilation. We finally explain the decisions and verify the properties of this tractable circuit using the techniques in (Shih, Choi, and Darwiche 2018b; 2018a).

This paper is organized as follows. In Section 2 we review 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. We provide a case study in Section 5, review related work in Section 6, and finally conclude in Section 7.

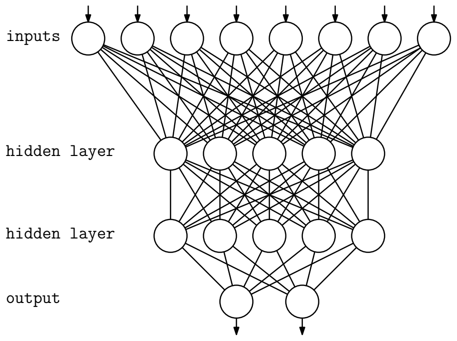
2 Technical Preliminaries

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.

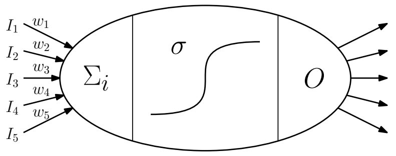
In this paper, we assume that the network inputs X_i are

[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).

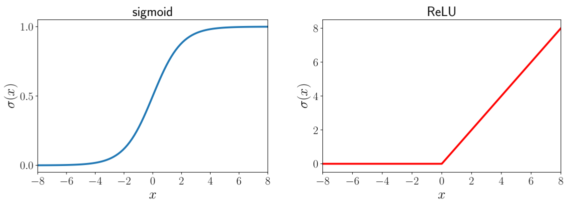
[2]See, for example, the EU general data protection regulation, which has a provision relating to explainability, `https://www.privacy-regulation.eu/en/r71.htm`.



(a) A neural network structure



(b) A mathematical model of a neuron



(c) Two activation functions

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

either 0 or 1. We further assume *step* activation functions:

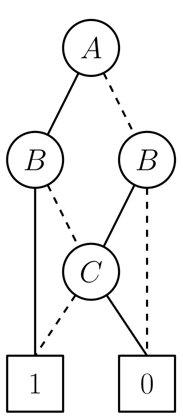
$$\sigma(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

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*.

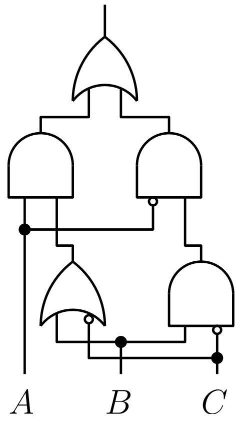
3 From Neural Networks to Boolean Circuits

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

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



(a) An OBDD



(b) A circuit

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

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 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

[3]An Ordered Binary Decision Diagram (OBDD) is a rooted DAG with two sinks: the 1-sink and 0-sink. An OBDD is a graphical representation of a Boolean function on variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Every OBDD node (except the sinks) is labeled with a variable X_i and has two labeled outgoing edges: the 1-edge and the 0-edge. The labeling of the OBDD nodes respects some 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 global 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.

Section 4. Note further that OBDDs are also circuits that are notated more compactly.[4]

Our goal now 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:*

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

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 a classifier's Boolean function.

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) provided an algorithm to obtain the result of Theorem 1, which has a much better average complexity than may be suggested by the bounds in the theorem. This algorithm was originally designed for compiling naive Bayes classifiers to Ordered Decision Diagrams (ODDs), but it applies to any classifier of the form given by Equation 2. This includes naive Bayes classifiers, but also logistic regression classifiers, as well as neurons with step activation functions; see also (Elkan 1997).

For completeness, we show next how a neuron with a step activation can be reduced to a naive Bayes classifier.

Proposition 1 *Consider a neuron with a step activation function σ, as in Equation 1. Let f denote the neuron's Boolean function. Consider the corresponding naive Bayes classifier, with binary attributes I_i (with values 0 and 1), and a class variable O (with values 0 and 1), with the CPTs:*

$$\theta_{O=1} = \frac{\exp\{\tau\}}{1+\exp\{\tau\}} \qquad \theta_{I=1|O=1} = \frac{1}{1+\exp\{-\frac{1}{2}w_i\}}$$

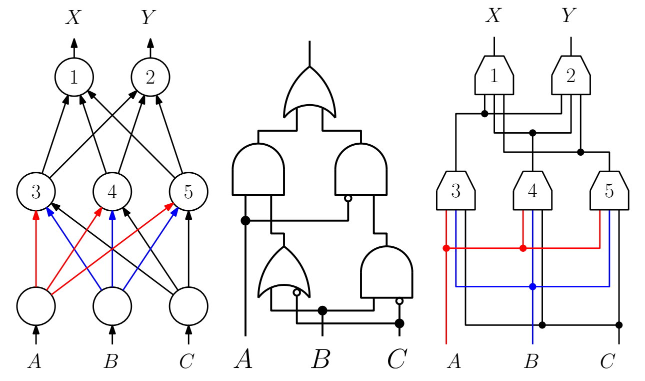
$$\tau = b + \frac{1}{2}\sum_i w_i \qquad \theta_{I=1|O=0} = \frac{1}{1+\exp\{\frac{1}{2}w_i\}}$$

Given any input $\mathbf{x}$, we have $f(\mathbf{x}) = 1$ iff $Pr(O=1 \mid \mathbf{x}) \geq \frac{1}{2}$.

The proof of this theorem is provided in the Appendix.

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

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



(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.

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 circuit that the neurons induce as a neural network is not necessarily tractable. However, the explanation and verification techniques proposed in (Shih, Choi, and Darwiche 2018b; 2018a), which we shall use, require a tractable circuit. We will next show how to obtain such a circuit using techniques from the literature on 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 mostly follow the conventions of (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 circuit 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 help motivate this trade-off, consider Figure 4, which highlights the containment relationship between four complexity classes. The "easiest" (smallest) class is NP, and the "hardest" (largest) class is $\mathrm{PP}^{\mathrm{PP}}$ (Oztok, Choi, and Darwiche 2016). 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 complete for that class, and to compile the resulting Boolean

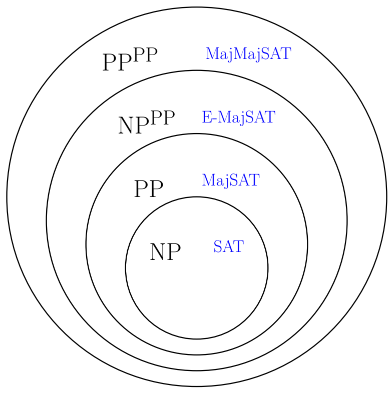


Figure 4: Containment of four complexity classes: $\mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{NP}^{\mathrm{PP}} \subseteq \mathrm{PP}^{\mathrm{PP}}$. The canonical problem that is complete for that class is labeled in blue.

expressions to Boolean circuits with the appropriate properties.[5] 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).[6] 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). 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 than OBDDs (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 $\alpha \vee \beta$.[7]

[5]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`

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

[7]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).

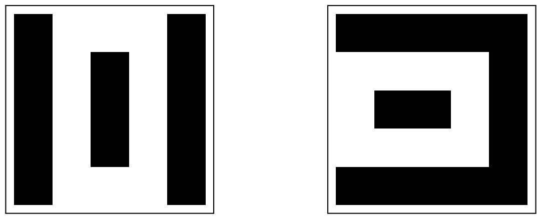


Figure 5: A tall rectangle (left) and wide rectangle (right).

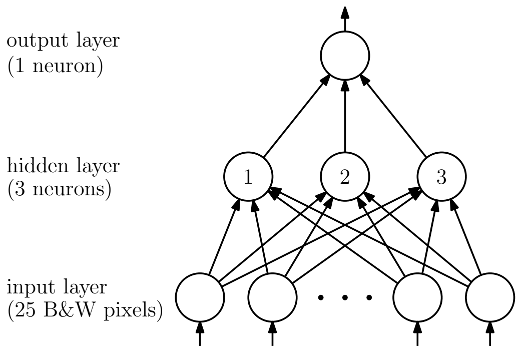


Figure 6: Architecture of the neural network analyzed.

Further, SDDs can be negated in linear time.[8]

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.[9]

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 A Case Study

We next provide a case study in explaining and verifying neural networks via knowledge compilation. We consider a

[8]The SDD package at `http://reasoning.cs.ucla.edu/sdd/` provides support for these operations, which we used in our case study in Section 5.

[9]In our experiments, we employed an alternative algorithm for compiling a circuit into an SDD. First, using the algorithm of (Chan and Darwiche 2003) as discussed earlier, for each neuron n we compile its Boolean function into an OBDD f_n. Next, we associate an auxiliary variable Z_n to the output of each neuron n. For each neuron, we assert the following equivalence constraint equating Z_n with f_n: $Z_n \Leftrightarrow f_n$ where $\alpha \Leftrightarrow \beta \equiv (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$. We then conjoin the equivalence constraints over all neurons: $\bigwedge_n (Z_n \Leftrightarrow f_n)$. Finally, we existentially quantify out the auxiliary variables Z_n and condition on output node Y. We then obtain a circuit over just the network inputs $\mathbf{X}$, which is logically equivalent to the compiled circuit described in the main text.

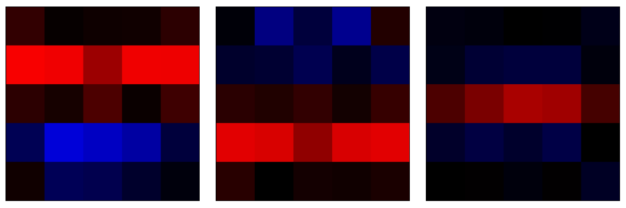

(a) first neuron (b) second neuron (c) third neuron

Figure 7: A visualization (using marginals) of the 3 neurons in the hidden layer.

synthetic image classification task: discriminating between tall and wide rectangles. We synthesize a binary dataset of 5×5 images containing either a tall rectangle (height greater than width) or a wide rectangle (width greater than height). Figure 5 depicts a tall rectangle with a height of 5 and a width of 3, and a wide rectangle with a height of 3 and a width of 4. White pixels represent the border of the rectangle; all other pixels are black. Each rectangle has a top and bottom row, and a left and right column, which are chosen randomly, but guaranteeing a height and width of at least 2 pixels. We give tall rectangles a 1-label (positive instance) and wide rectangles a 0-label (negative instance). Squares (equal height and width) are given 1-labels. In a 5×5 grid, there are 100 unique rectangles.

We trained a simple two-layer feedforward neural network using TensorFlow; see Figure 6. We assumed 3 nodes in the hidden layer, and minimized a cross entropy loss using the Adam optimizer with a learning rate of 0.1, for 500 epochs. We sought a neural network that facilitated explanation and verification (as we shall see shortly). Hence, we use all 100 unique examples for training, and trained with different seeds until we obtained a simple network with 100% training set accuracy.[10] To compile a neuron to an OBDD, using the algorithm of (Chan and Darwiche 2003), we need to assume step activations. However, the step activation function is not differentiable at zero, and has a zero derivative everywhere else. In practice, one can train the network with sigmoid activations and then assume step activations at test time. We do the same here. Further, when using step activations, we also achieved 100% accuracy on the training set. After compiling all neurons to OBDDs/SDDs, the neural network's circuit had an (aggregate) size of 15,073, although it is not yet tractable.[11] After compiling this circuit to a tractable circuit, i.e., an SDD, as described in the previous section, the circuit size was 92,998 (the increased size is the trade-off for obtaining tractability).

Interpreting Neurons via Model Counting: In the process of compiling a neural network into a tractable circuit (an SDD), we also compile each neuron into a tractable circuit, as a function of the network's inputs. Hence, we first propose an approach for interpreting each neuron using its tractable circuit, through a visual analysis. Consider the following question: Of all network inputs that cause a neuron to fire (i.e., has output 1), what proportion of them set input X_i to 1? Or in other words, given that the circuit output is 1 what is the marginal probability that the input X_i is 1 (assuming a uniform distribution over inputs)? By visualizing these marginals across all pixels, this gives us a sense of what types of inputs will cause a neuron n to fire. Given an SDD, we can answer the proposed question through *model counting*.[12] The model count tells us how many input vectors will cause a high output to a circuit. Given an SDD, its model count can be computed in time *linear* in the size of the circuit. Note that model counting is a query that is beyond the capabilities of approaches to verification based on satisfiability (model counting is a #P-complete problem); we discuss this point further in Section 6.

Figure 7 depicts the marginals for each neuron in the hidden layer of the neural network that we learned. 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. Consider for example Figures 7a & 7b. Given that the first neuron fired, it is highly probable that pixels on the second row were white and that the pixels on the fourth row were black. Similarly, for the second neuron, except that the role of the rows are swapped. One or both neurons will fire on wide rectangles, depending on the alignment of its horizontal borders. The third neuron in Figure 7c would also tend to fire on wide rectangles that have a border on the middle row, although its behavior is a bit more subtle, as we shall discuss shortly.

Consider the output neuron. By examining its Boolean function, we can explain the behavior of the neural network as a function of the neurons in its hidden layer. The output neuron has the following form:

$$(-14.225 \cdot A) + (-13.573 \cdot B) + (6.421 \cdot C) + 1.186 \geq 0$$

where inputs A, B and C are the outputs of the three neurons in the hidden layer. We see that the first and second neurons contribute negatively to the decision of a tall rectangle, while the third neuron surprisingly has a positive impact. In fact, *all* tall rectangles in the dataset cause the third neuron to fire.[13] However, we find that the third neuron has *no role on the output of the neural network*. If we consider the SDD circuit of the output neuron, and the logical expression that it represents, we see that it fires iff $\neg A \wedge \neg B$.

[10] A neural network with just 3 nodes in the hidden layer can obtain 100% training accuracy without "memorizing" the dataset. As we shall see in this case study, we can analyze the network and verify that it is learning some more general properties of the dataset. Note further that not all neural networks that we learned from this dataset were as clear as the one that we analyze here.

[11] The size of an SDD node is the number of its children. The size of an SDD is the aggregate size of its nodes.

[12] Let Δ_n be the SDD representing neuron n. The model count of Δ_n is the number of its satisfying assignments, i.e., the number of inputs leading to a high output. If $\mathsf{mc}(\Delta_n)$ is the model count of Δ_n, and $\mathsf{mc}(X_i \wedge \Delta_n)$ is the model count of Δ_n when X_i is set to 1, then $Pr_{\Delta_n}(X_i) = \mathsf{mc}(X_i \wedge \Delta_n)/\mathsf{mc}(\Delta_n)$ is the proportion of the models of Δ_n where X_i is set to 1.

[13] To see why, consider Figure 7c. When this neuron fires, the pixels in the middle row are likely to be white. However, consider the fact that the minimum width of a rectangle is two, and so the minimum height of a tall rectangle must be three. Since our images are 5×5 pixels, any tall rectangle must cross at least two pixels from the middle row, hence causing this neuron to fire.

That is, an image is classified as a tall rectangle iff the first two neurons do not fire. Equivalently, an image is classified as a wide rectangle iff at least one of the first two neurons fire. More importantly, we see that the neural network's output is independent of the third neuron, and hence it can be dropped without affecting the behavior of the network. That is, to discriminate between tall and wide rectangles, it suffices to consider just the first two neurons. Note that detecting unused inputs in a circuit is another efficient operation on tractable circuits (Shih, Choi, and Darwiche 2018a).[14] As our case study highlights, tractable circuits lend themselves to pruning neural networks as well (i.e., to optimize their structure for memory and speed).

Explaining a Decision: Next, 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). Say that input image $\mathbf{x}$ is classified positively, i.e., a tall rectangle. A PI-explanation returns the shortest 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 over pixels do not matter—the network will always classify the instance as a tall rectangle.

First, consider the correctly classified tall rectangle in Figure 8a, and its PI-explanation in Figure 8b. For the neural network to classify this image as a tall rectangle, it suffices to see white pixels on three of its corners, and a few black pixels on both sides—the other pixel values do not matter. With the settings of these pixels, any completion to a rectangle will result in one that has a width of at most 3 and a height of at least 4. In Figure 8c & 8d, we see another tall rectangle and its PI-explanation. We can make similar observations here (remember that squares are included in the tall class).

Figures 8e & 8f and Figures 8g & 8h consider two wide rectangles and their PI-explanations. This time, consider the PI-explanation in Figure 8f and the visualization of the neuron in Figure 7b, which is replicated in Figure 8j. We see that the pixels set to white and black by the PI-explanation are also likely to be set to white and black when the second neuron of the hidden layer fires. Further, we know that the network predicts a rectangle iff at least one of the first two neurons fire. From this, we can infer that the rectangle in Figure 8e is being classified as wide because the PI-explanation of Figure 8f is forcing the second neuron to fire. We can infer similarly that the rectangle in Figure 8g is being classified as wide because the PI-explanation of Figure 8h is forcing the first neuron of Figure 7a (and 8i) to fire.

Interpreting the Neural Network Output: Figure 9a highlights the marginals of the output neuron, i.e., the probability that each pixel is white given that the output of the network is "tall." A renormalized version is given in Figure 9h. Perhaps not surprisingly, we find that if the output of the neural network is high, then it is somewhat more likely

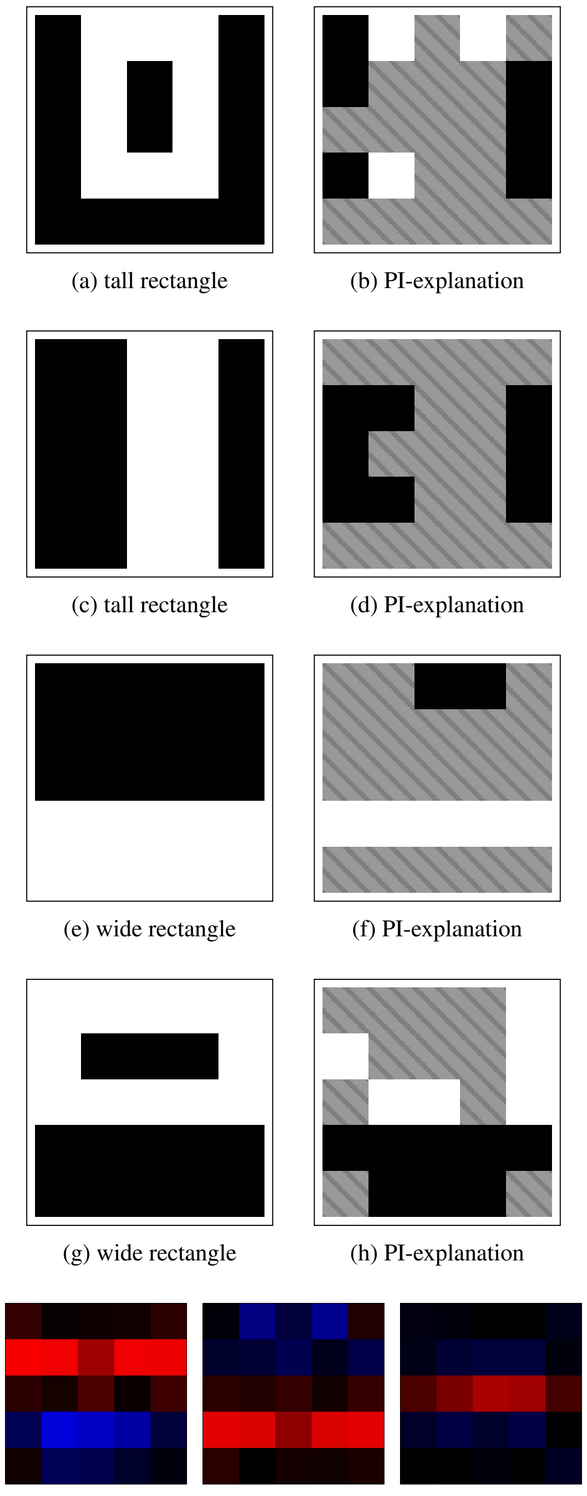


Figure 8: Two correctly classified tall rectangles (8a & 8c), and two correctly classified wide rectangles (8e & 8g). For convenience, we replicate the visualizations of the hidden neurons of Figure 7 (8i-8k).

[14]In general, determining whether the input of a neuron is unused is an NP-hard problem. This can be shown using a reduction similar to (Shih, Choi, and Darwiche 2018b), which showed that compiling a naive Bayes classifier is an NP-hard problem.

that the three central pixels in the top and bottom rows are set to white. Further, the three central pixels in the left and right columns are somewhat more likely to be set to black. These are properties suggestive of a tall rectangle. Next, we shall take a closer look at the behavior of the neural network.

Consider again the PI-explanation of a *wide* rectangle in Figure 8f. In Figure 9b through Figure 9g, we consider the conditional probability of each input pixel being white given that we incrementally set the pixels of the PI-explanation. We want to observe how the neural network reacts in this case: the more pixels of the PI-explanation that we set, the more difficult it is for the neural network to predict a tall rectangle. Remember, that if we set all pixels of this PI-explanation, then the neural network can no longer predict "tall." Hence, we do not show the result of setting the full PI-explanation.

First, in Figure 9b, we have set a single pixel of the PI-explanation. The resulting conditional probabilities resemble the original marginals of Figure 9a. As we set more pixels of the PI-explanation, we see the conditional probabilities becoming more extreme. As we move right in Figure 9, the marginals more strongly resemble an inverted version of Figure 7b (replicated in Figure 9j), which visualized the marginals of the second neuron. Remember that in this network, to classify an input image as a tall rectangle, neither the first nor the second neuron of Figure 7 are allowed to fire. We see this reflected in Figure 9: as we move left-to-right, in order to keep classifying the input image as tall, we have to do more to prevent the second neuron from firing (i.e., set more inputs that keep the neuron below its threshold).

We emphasize a few points now. First, this (visual) analysis is enabled by the tractability of the circuit, which allows model counts and marginals to be computed efficiently. Second, the analysis also emphasizes that the neural network is not learning "rectangles," per se. It learned some properties of rectangles (i.e., the first two neurons of the hidden layer), that allowed the network to classify them with 100% accuracy. This perhaps explains why it is sometimes easy to "fool" neural networks (e.g., in Figure 9g, it is still possible for the neural network to classify some of its input as tall, even though it is no longer possible to extend the 6 pixels already set to a tall rectangle). Finally, while our case study concerned a very small neural network by today's standards, our findings clearly show the promise of the proposed direction of research as it gives a sense of the insights one can gain by compiling the Boolean functions of neural networks intro tractable circuit representations. Scaling this compilation approach to more realistic neural networks is a current focus of ours.

6 Related Work

Our approach follows a recent trend in analyzing machine learning models using symbolic approaches such as satisfiability (SAT) and satisfiability modulo theory (SMT); see, e.g., (Katz et al. 2017; Leofante et al. 2018; Narodytska et al. 2018; Cheng et al. 2018; Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019). While machine learning and statistical methods are key for learning classifiers, it is evident 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, is capable of going beyond queries based on (for example) satisfiability (NP–complete) and equivalence checking (co-NP–complete), to problems beyond NP such as model counting, MajSAT and MajMajSAT (Oztok, Choi, and Darwiche 2016; Choi, Darwiche, and Van den Broeck 2017). Access to such queries can provide more sophisticated approaches to explanation and verification.

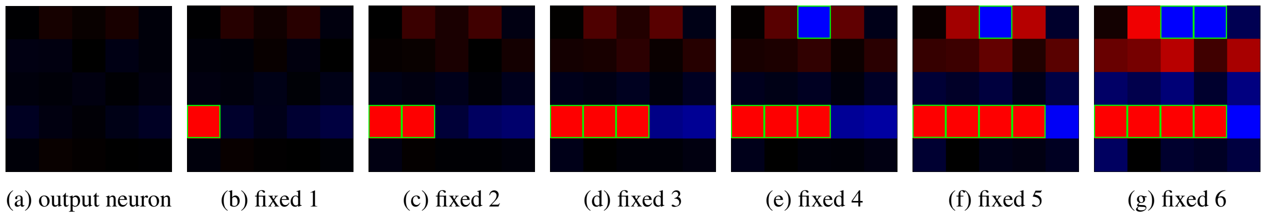
In parallel with this paper, (Shih, Choi, and Darwiche 2019) propose another knowledge compilation approach for verifying neural networks. In particular, their approach is based on *learning* the OBDD of a neural network's Boolean function. The approach is based on algorithms for learning deterministic finite state automata (dfsa) (Angluin 1987), using membership and equivalence queries that are made to an oracle. In this case, membership queries can be made on the original neural network. Equivalence queries (between the original neural network and the candidate learned model) can be made by reducing both the neural network and candidate OBDD to a joint CNF instance, where a SAT solver can now be used to test equivalence (Narodytska et al. 2018). There are a number of differences with this work. First, they consider a different class of neural networks called Binarized Neural Networks (BNNs) (Hubara et al. 2016; Narodytska et al. 2018).[15] Next, our approach to compilation targets the more general class of SDDs, is conceptually simpler, and also more general. On the other hand, (Shih, Choi, and Darwiche 2019) inherits certain guarantees from the original dfsa learning algorithm.[16] Finally, the approach proposed by (Shih, Choi, and Darwiche 2019) is concerned with compiling the function of a neural network in a region "around" a given instance, whereas our approach seeks to compile the whole Boolean function of a neural network. While a region-based approach restricts the types analyses that are possible (explanation and verification), it can be more scalable if the region is small enough.

7 Conclusion

In this paper, 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 represents how a neural network labels an input feature vector with a class. We showed how to compile the Boolean function of each neuron and the network itself into a tractable circuit, and into an Sentential Decision Diagram (SDD) in particular. In a case study, we showed how polytime queries supported by the SDD can be utilized to efficiently explain and verify the behavior of neural networks.

[15]The "binarized" in a BNN refers to its binary parameters and activations, and not just binary inputs like we assume in this paper.

[16]In particular, the algorithm requires a number of equivalence queries that is linear in the size of the output OBDD, where equivalence checking is the primary bottleneck of the procedure.



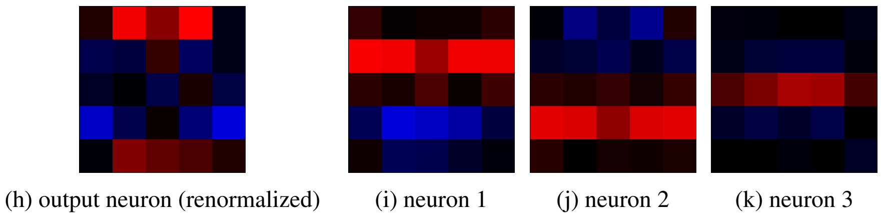


Figure 9: A visualization of the output neuron (9a), and after conditioning on the pixels of the PI-explanation of Figure 8f, one pixel at a time from left-to-right (9b–9g). The fixed pixels are outlined in green. A renormalized visualization of the output is given in 9h (the magnitudes of the probabilities are rescaled to be closer to 0 and 1). For convenience, we replicate the visualizations of the hidden neurons of Figure 7 (9i-9k).

Acknowledgments We thank Yujia Shen for comments and discussions on this paper. This work has been partially supported by NSF grant #IIS-1514253, ONR grant #N00014-18-1-2561 and DARPA XAI grant #N66001-17-2-4032.

A Proofs

Proof of Proposition 1 Let $\mathbf{i}$ denote an input vector. Given activation function σ a neuron first computes:

$$\begin{aligned}\sum_i w_i \cdot I_i + b &= b + \sum_i w_i \cdot I_i + 0 \cdot (1 - I_i) \\ &= b + \sum_i \frac{1}{2} w_i \cdot I_i - \frac{1}{2} w_i \cdot (1 - I_i) + \frac{1}{2} w_i \\ &= \left(b + \frac{1}{2} \sum_i w_i \right) + \sum_i \frac{1}{2} w_i \cdot I_i - \frac{1}{2} w_i \cdot (1 - I_i)\end{aligned}$$

We then have the log-odds:

$$\begin{aligned}\log \frac{Pr(O=1 \mid \mathbf{i})}{Pr(O=0 \mid \mathbf{i})} &= \log \frac{Pr(O=1)}{Pr(O=0)} \frac{Pr(\mathbf{i} \mid O=1)}{Pr(\mathbf{i} \mid O=0)} \\ &= \log \frac{Pr(O=1)}{Pr(O=0)} + \sum_i \log \frac{Pr(I_i \mid O=1)}{Pr(I_i \mid O=0)}\end{aligned}$$

If $I_i = 1$ then

$$\begin{aligned}\log \frac{Pr(I_i=1 \mid O=1)}{Pr(I_i=1 \mid O=0)} &= \log \frac{1 + \exp\{\frac{1}{2} w_i\}}{1 + \exp\{-\frac{1}{2} w_i\}} \\ &= \log \frac{\exp\{\frac{1}{2} w_i\} \cdot (1 + \exp\{\frac{1}{2} w_i\})}{1 + \exp\{\frac{1}{2} w_i\}} = \frac{1}{2} w_i.\end{aligned}$$

If $I_i = 0$ then

$$\begin{aligned}&\log \frac{Pr(I_i=0 \mid O=1)}{Pr(I_i=0 \mid O=0)} \\ &= \log \frac{\exp\{-\frac{1}{2} w_i\}}{\exp\{\frac{1}{2} w_i\}} \frac{1 + \exp\{\frac{1}{2} w_i\}}{1 + \exp\{-\frac{1}{2} w_i\}} = -\frac{1}{2} w_i.\end{aligned}$$

Further, $\log \frac{Pr(O=1)}{Pr(O=0)} = \log \exp\{\tau\} = \tau$ and hence,

$$\log \frac{Pr(O=1 \mid \mathbf{i})}{Pr(O=0 \mid \mathbf{i})} = \tau + \sum_i \frac{1}{2} w_i \cdot I_i - \frac{1}{2} w_i \cdot (1 - I_i).$$

Thus, $\sum_i w_i \cdot I_i + b \geq 0$ iff $\log \frac{Pr(O=1 \mid \mathbf{i})}{Pr(O=0 \mid \mathbf{i})} \geq 0$. $\square$

References

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75(2):87–106.

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.

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.

Cheng, C.; Nührenberg, G.; Huang, C.; and Ruess, H. 2018. Verification of binarized neural networks via inter-neuron factoring (short paper). In *Proceedings of the 10th International Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, 279–290.

Choi, Y.; Darwiche, A.; and Van den Broeck, G. 2017. Optimal feature selection for decision robustness in Bayesian networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*.

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.

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.

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

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.

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 $\mathrm{PP}^{\mathrm{PP}}$-complete problems using knowledge compilation. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 94–103.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "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.; Choi, A.; and Darwiche, A. 2019. Verifying binarized neural networks by local automaton learning. In *AAAI Spring Symposium on Verification of Neural Networks (VNN)*.

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

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.