On Bounding the Behavior of Neurons

Richard Borowski **and** Arthur Choi
Computer Science Department
College of Computing and Software Engineering
Kennesaw State University
rborowsk@students.kennesaw.edu, achoi13@kennesaw.edu

April 26, 2024

Abstract

A neuron with binary inputs and a binary output represents a Boolean function. Our goal is to extract this Boolean function into a tractable representation that will facilitate the explanation and formal verification of a neuron's behavior. Unfortunately, extracting a neuron's Boolean function is in general an NP-hard problem. However, it was recently shown that prime implicants of this Boolean function can be enumerated efficiently, with only polynomial time delay. Building on this result, we first propose a best-first search algorithm that is able to incrementally tighten the inner and outer bounds of a neuron's Boolean function. Second, we show that these bounds correspond to truncated prime-implicant covers of the Boolean function. Next, we show how these bounds can be propagated in an elementary class of neural networks. Finally, we provide case studies that highlight our ability to bound the behavior of neurons.

1 Introduction

Rapid advances in artificial intelligence, and its increasing pervasiveness, has brought with it the need to understand and explain the behavior of the resulting systems. This need gave rise to a new sub-field of AI, called eXplainable Artificial Intelligence (XAI).[1–4] *Formal* approaches to XAI, in particular, seek to provide formal guarantees on the behavior of such systems, e.g., by providing bounds on the output of a neural network (say, a guarantee that a self-driving car does not exceed safe driving speeds).[5–11]

Unfortunately, for a sufficiently powerful notion of explanation, it is NP-hard to explain the behavior of a neural network.[11] For example, deciding if a neural network ever produces a positive labeling is analogous to testing the satisfiability of a Boolean formula. Even worse: it is NP-hard to explain the behavior of an *individual neuron.* It is NP-hard to decide if a neuron outputs a 1 more often than it outputs a 0. It is also NP-hard to compile an individual neuron into a more tractable representation,[7] such as an Ordered Binary Decision Diagram (OBDD).[1]

Fortunately, there is recourse to this apparent intractability. For example, a neuron (with a step-activation) will admit a pseudo-polynomial time compilation into an OBDD if its weights are integers. Further, if the aggregate weight of such a neuron is bounded, then it can be compiled into an OBDD in polytime.[16–18] More recently, Marques-Silva et al. showed that prime implicants (PIs) can be efficiently enumerated from a linear classifier.[19] Prime implicants, and the corresponding PI-explanations (or sufficient explanations), provide a partial description of the behavior of a classifier.[7, 9, 20, 21] In the same way that a Boolean function can be decomposed according to its prime implicants, the behavior of a neuron can be decomposed according to its PI-explanations.

In this paper, our goal is to obtain inner and outer bounds on the behavior of simple neural networks. We start with the results of Marques-Silva et al.,[19] on efficiently enumerating prime implicants from linear classifiers. By correspondingly enumerating prime implicants from a neuron's Boolean function, we can obtain inner and outer bounds on the behavior of a neuron. Our first contribution is to formulate this enumeration problem as a best-first search, giving us the ability to obtain inner and outer bounds that are much tighter that what we could obtain before. Subsequently, this best-first search yields a polytime approximation of a neuron's Boolean function, which we formally characterize as a truncated prime implicant cover. Towards the longer-term goal of bounding the behavior of deep neural networks, our next contribution is to show how these inner and outer bounds can be propagated through a network of neurons, providing inner and outer bounds on an elementary class of neural networks, hence, going beyond the scope of Marques-Silva et al.[19] Empirically, through two case studies, we show how our algorithm is able to provide near-total coverage of the behavior of neurons, by enumerating a relatively small number of prime implicants.

This paper is organized as follows. First, in Section 2, we characterize the behavior of a neuron in simpler terms, as a threshold test. In Section 3, we show how the behavior of a threshold test can be bounded by prime implicants. Next, in Section 4, we define a search space over threshold tests, which we explore via best-first search, to tighten inner and outer bounds on the behavior of a threshold test. In Section 5, we consider how to propagate these inner and outer bounds through an elementary class of neural networks. In Section 6, we work out a simple example of our proposed bounds using a small neural network. We provide two case studies that highlight our ability to bound the behavior of neurons, in Section 7. Finally, we conclude in Section 9.

[1]An OBDD is a *tractable* representation of a Boolean function that supports polynomial time transformations and operations,[12–14] which facilitate the explanation and formal verification of a neuron.[8, 10] OBDDs are studied in the field of knowledge compilation, a sub-field of AI that studies in part tractable representations of Boolean functions, and the trade-offs between their succinctness and tractability.[15]

2 On Neurons as Threshold Tests

Consider *binary* neurons with

1. binary inputs $I_1, \ldots, I_n$ that are 0 or 1, and
2. a binary output that is 0 or 1.

Further, assume that the neuron has a step activation $\sigma(x) = 1$ if $x \geq 0$ and $\sigma(x) = 0$ otherwise. Such a neuron has the form:

$$f(I_1, \ldots, I_n) = \sigma\left(w_1 I_1 + w_2 I_2 + \cdots + w_n I_n + b\right)$$

where w_i is the weight on input I_i, and b is a bias. Such a neuron can be viewed as a function mapping binary inputs to a binary output, i.e., a Boolean function. We refer to this as the *neuron's Boolean function.*

Some binary classifiers, including neurons with step activations, can be viewed more generally as a *threshold test.*

Definition 1. *A* threshold test f *is a function with* n *inputs* $I_1, \ldots, I_n$ *that are 0 or 1, with weights* $w_1, \ldots, w_n$ *and a threshold* T*. The output of a threshold test is 1 iff*

$$w_1 I_1 + w_2 I_2 + ... + w_n I_n \geq T$$

and we say that the test passes*. Otherwise, the output is 0 and we say that the test* fails*.*

Note that a negated threshold $-T$ is a bias b in a neuron.

Remark 1. *The output of a binary neuron is 1 iff the corresponding threshold test passes.*

Consider, as a running example, the following threshold test:

$$3 \cdot I_1 + 2 \cdot I_2 - 4 \cdot I_3 \geq 1. \tag{1}$$

We can enumerate all possible inputs and record the corresponding output, as we would in a truth table:

I_1	I_2	I_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0

I_1	I_2	I_3	f
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

.

We say that a threshold test f *always passes* iff the left-hand side is always greater than or equal to the threshold, no matter how we set the inputs. Similarly, we say that a threshold test f *always fails* iff the left-hand side is always less than the threshold. We call a threshold test *reduced* or *trivial* if it either always passes or it always fails.

In our example, if we set input I_1 to 0 and input I_2 to 0, then the resulting threshold test has been reduced and always fails, no matter how we set input I_3:

$$-4 \cdot I_3 \geq 1. \tag{2}$$

Suppose instead that we set input I_1 to 1 and input I_2 to 1. After subtracting 5 from both sides, the resulting threshold test has been reduced and always passes:

$$-4 \cdot I_3 \geq -4. \tag{3}$$

Observe that the left-hand side of a threshold test is minimized by setting all inputs with positive weight to 0 and all inputs with negative weight to 1. Similarly, the left-hand side is maximized by setting all inputs with positive weight to 1 and all inputs with negative weight to 0. In our example, the left-hand side has a minimum of -4 and a maximum of 5.

Definition 2. *Suppose we have a threshold test f, where we let W^+ denote the set of positive weights and we let W^- denote the set of negative weights. The threshold test f has a* lower bound *L and* upper bound *U where:*

$$L = \sum_{w \in W^-} w \qquad\qquad U = \sum_{w \in W^+} w.$$

The range *of a threshold test is thus $[L, U]$ where:*

$$L \leq w_1 I_1 + w_2 I_2 + \cdots + w_n I_n \leq U.$$

for all settings of $I_1, \ldots, I_n$ to 0/1 values.

This leads to a simple condition for testing whether a threshold-test always passes, or always fails.

Proposition 1. *Let f be a threshold test with threshold T and range $[L, U]$.*

- *A threshold test f always passes iff $T \leq L$.*
- *A threshold test f always fails iff $U \leq T$.*

The original threshold test of Equation 1 has a range $[-4, 5]$ and a threshold 1 and is not yet reduced. The threshold test of Equation 2 has a range $[-4, 0]$ and a threshold 1 and thus always fails. The threshold test of Equation 3 has a range $[-4, 0]$ and a threshold -4 and thus always passes.

3 Bounding the Behavior of a Threshold Test

The function f representing a threshold test outputs a 1 if the threshold test passes, and outputs a 0 otherwise. Consider a partial setting of the inputs α that reduces a threshold test into one that always passes. This α is a more concise description of the behavior of the threshold test, in that it summarizes many input settings that cause the threshold test to pass (exponentially many, in the number of unset inputs).

In logical terms, a (partial) setting of inputs is a conjunction of literals, which we call a *term.* A term α is also called an *implicant* of a Boolean function f if α entails f, i.e., each extension of a partial setting α to a total setting, over all inputs, results in an assignment satisfying f. We call α a *prime implicant* if no sub-term of α is also an implicant. A *prime cover* is a decomposition of a function f into prime implicants.[22] The prime cover of a function f may not be unique, and we generally prefer *irredundant* covers that contain fewer implicants. Suppose that a prime cover of f has m implicants α_i:

$$f = \alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_m$$

Note that each implicant α_i summarizes a sub-space of the inputs that satisfy f. These implicants in aggregate provide a precise description of the behavior of f. Suppose that we have a subset A of a prime cover of f and a subset B of a prime cover of $\neg f$. Such subsets (or truncations) of a prime cover yield inner and outer bounds on the behavior of f:

$$\bigvee_{\alpha \in A} \alpha \quad \models \quad f \quad \models \quad \bigwedge_{\beta \in B} \neg\beta. \tag{4}$$

Hence, if we can enumerate the prime implicants of a threshold test's Boolean function, we can tighten inner and outer bounds on its behavior.

Marques-Silva et al.[19] showed that prime implicants can be efficiently enumerated for a broad class of linear classifiers, including threshold tests.

Theorem 1. *The prime implicants of a linear classifier's Boolean function can be enumerated with polynomial delay.*

Proof. See Ref. 19. □

Next, we provide a simplified perspective on this result, based on best-first search, where we enumerate the shortest (most informative) prime implicants first.

4 Enumerating the Space of Threshold Tests

If we fix the input of a threshold test to a value, we obtain a simpler threshold test with one fewer input. This induces a space of threshold tests based on setting inputs to values. Figure 1 depicts an example search tree, where each node represents a threshold test, and where each directed edge $f \rightarrow g$ represents the setting of an input in threshold test f to obtain threshold test g. For example, we have the following threshold test at the root of this tree:

$$-4 \cdot I_1 - 2 \cdot I_2 + 3 \cdot I_3 + 5 \cdot I_4 \geq 3.$$

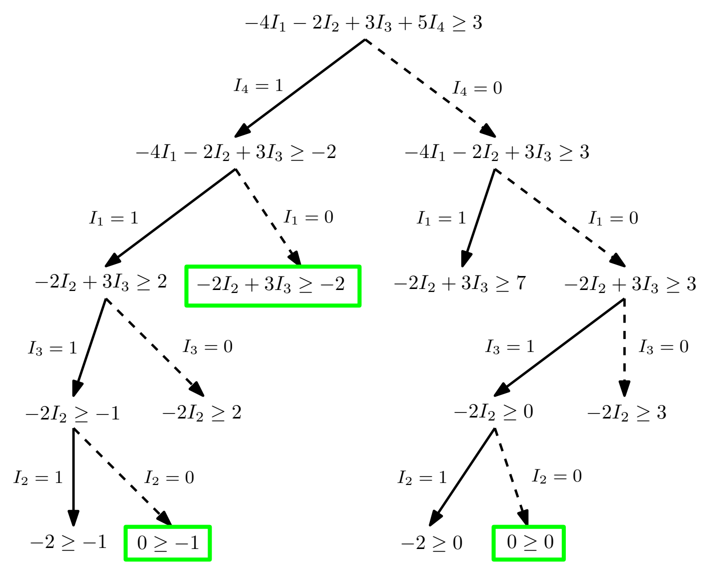


Figure 1: Decision tree over threshold tests.

If we set input I_4 to 0, we obtain the simpler threshold test:

$$-4 \cdot I_1 - 2 \cdot I_2 + 3 \cdot I_3 \geq 3$$

by following the dashed edge. If we instead set input I_4 to 1, we obtain the threshold test by following the solid edge:

$$-4 \cdot I_1 - 2 \cdot I_2 + 3 \cdot I_3 \geq -2$$

after subtracting 5 from both sides. We can continue to expand the tree until we have set all inputs to values, where we have left a comparison between two constants, which either passes or fails. For example, if we set all inputs to one (by following all of the solid lines from the root), we obtain the comparison $-2 \geq -1$ which fails.

By expanding all such paths, we obtain a decision tree representing the threshold test's Boolean function. Note that once a threshold test reduces to a trivial one, we need not expand its sub-tree, as all of its leaves will have the same status (either all passing, or all failing). For example, after setting I_4 to 1, and I_1 to 0 in the root threshold test we have:

$$-2 \cdot I_2 + 3 \cdot I_3 \geq -2$$

which has a range $[-2, 3]$ and a threshold -2 and is thus always passing. Thus, we can prune our decision tree by not expanding sub-trees rooted at trivial threshold tests.

Consider the paths from the root to each leaf of a threshold test's decision tree (pruned or not). Each path is a conjunction of literals (a term) composed of the input settings on the path. The decision tree's Boolean function is the disjunction of the path terms to the always-passing leaves.

A shallower decision tree has fewer leaves with shorter path terms, and hence a shallower decision tree represents a more compact representation of a threshold test's Boolean function. Our next goal is to find a shallow decision tree. Subsequently, we will also show how the decision tree represents a prime cover of the threshold test's Boolean function.

Suppose, more formally, that we have a threshold test f with inputs $I_1, \ldots, I_n$, weights $w_1, \ldots, w_n$, a threshold T and range $[L, U]$. When we fix the value of an input I that has a corresponding weight w, we obtain a simpler threshold test that has (1) one fewer input, (2) an updated threshold, and (3) an updated range. We have four cases, based on the sign of the weight and the value that we set the input to:

- if $w \geq 0$ and $I{=}0$ then we have updated range $[L, U - w]$ and threshold T
- if $w \geq 0$ and $I{=}1$ then we have updated range $[L, U - w]$ and threshold $T - w$
- if $w < 0$ and $I{=}0$ then we have updated range $[L - w, U]$ and threshold T
- if $w < 0$ and $I{=}1$ then we have updated range $[L - w, U]$ and threshold $T - w$

We can also quantify how close we are to reducing a threshold test.

Definition 3. *Say we have a threshold test f with threshold T and range $[L, U]$.*

- *If the threshold test is not always passing, then we have that $L < T$ and the* gap *before the test becomes always passing is $T - L$.*
- *If the threshold test is not always failing, then we have that $T \leq U$ and the* gap *before the test becomes always failing is $U - T$.*

A setting of an input is called reducing *if it reduces the gap.*

To close the gap of a threshold test towards always passing, we can set an input with a negative weight $-w$ to 0 to raise the lower bound L, or we can set an input with a positive weight w to 1 to lower the threshold T. To close the gap of a threshold test towards always failing, we can set an input with a positive weight w to 0 to lower the upper bound U, or we can set an input with a negative weight $-w$ to 1 to raise the threshold T. In all cases, the gap decreases by w. Hence, to reduce a threshold test to a trivial one, it suffices to set inputs to values that close the corresponding gap. To close this gap *quickly*, by setting the fewest inputs, it follows from Marques-Silva et al.[19] that a greedy approach suffices. That is, we pick the smallest set of inputs whose aggregate (absolute) weight meets or exceeds the gap.

Proposition 2. *Suppose we have a threshold test f that is not yet reduced, and let G be the gap until it becomes always passing (or always failing). To reduce f to a trivial threshold test, by setting the fewest number of inputs, iteratively pick the input I_i with largest absolute weight $|w_i|$ and set the input to its reducing value, until the gap is closed.*

Proof. See Ref. 19. □

Consider the test at the root of the decision tree in Figure 1:

$$-4 \cdot I_1 - 2 \cdot I_2 + 3 \cdot I_3 + 5 \cdot I_4 \geq 3.$$

which has range $[-6, 8]$ and threshold 3. The gap until it always passes is $T - L = 3 - (-6) = 9$, and the fastest way to reduce it is to set I_4 to 1 and I_1 to 0. The gap until it always fails is $U - T = 8 - 3 = 5$, and the fastest way to reduce it is to set I_4 to 0 and I_1 to 1 (note that to reduce it to always failing, we must strictly clear the gap, as the test passes if the left-hand side is still equal to the threshold).

4.1 Search in the Decision Tree

As discussed, a shallower decision tree is a more compact representation of a threshold test's Boolean function. Once we have fixed the decision tree, we also want to enumerate its leaves from shallowest to deepest, as shallower leaves have shorter paths that are more informative.

We propose to answer both points by formulating the problem as a best-first search (such as A* search) in the decision tree. Initially, we have a priority queue containing just the initial threshold test, i.e., the root of the decision tree. The goal states in our search are the leaf nodes that correspond to always-passing threshold tests. The priority queue ranks threshold tests based on the number of inputs that need to be set to reduce it to a trivial one; each threshold test can be scored in polytime using Proposition 2. In each iteration, we pop the threshold test needing the fewest inputs set to reduce it. We find the unset input I with largest absolute weight $|w|$, as in Proposition 2. We produce two simpler threshold tests found by setting input I to 0 and 1, which we push back into the priority queue, and go on to the next iteration.

The first goal node that we find will be the shallowest leaf node in the decision tree, which corresponds to the shortest prime implicant of the decision tree's Boolean function.[19] We can continue the search to enumerate the next shallowest leaf node, and the next shallowest leaf node, and so on until we enumerate all leaf nodes (and we have covered the entire function), or until we consume the computational resources available to us (and we have only a bound on the function).

4.2 A Decision Tree is a Prime Cover

Let f denote a decision tree's Boolean function, which is found by disjoining all paths to its always-passing leaves, where $\neg f$ is the function found by disjoining all paths to the always-failing leaves. If the decision tree is not pruned, then this representation simply enumerates all models (or satisfying assignments) of f.

In a pruned decision tree, consider a path α to a leaf representing an always-passing threshold test. This path α is an implicant of the function f: any completion of α to all inputs results in a passing threshold test, and thus corresponds to a satisfying assignment of f. Hence, a pruned decision tree represents a decomposition of the Boolean function f into implicants. For each path α, we can always obtain from it a *prime implicant*, using the following Lemma.

Lemma 1. *Say we have a threshold test f and a partial assignment γ of its inputs, where input I is non-reducing. If γ is an implicant of f, then $\gamma \setminus I$ is also an implicant of f.*

Proof. Say we have threshold test $w_1 I_1 + \cdots + w_n I_n \geq T$, its Boolean function f, and an implicant γ of f. Suppose input I_k was a non-reducing setting in γ. Since γ is an implicant, setting γ results in an always-true threshold test where $T_\gamma \leq [L_\gamma, U_\gamma]$. We want to show that $\kappa = \gamma \setminus I_k$ remains an implicant, and the threshold test is still always-true. Consider two cases. (1) If $w_k > 0$ then setting I_k to 0 is non-reducing. If we unset I_k, then κ induces another test with threshold $T_\kappa = T_\gamma$ and range $L_\kappa = L_\gamma$ and $U_\kappa = U_\gamma - w_I$. Since $T_\gamma \leq L_\gamma \leq U_\gamma$, we have $T_\kappa \leq L_\kappa \leq U_\kappa$, so the threshold test is still trivial. (2) If $w_k < 0$ then setting I_k to 1 is non-reducing. If we unset I_k, then κ has a test with threshold $T_\kappa = T_\gamma - w_k$ and range $L_\kappa = L_\gamma - w_k$ and $U_\kappa = U_\gamma$. Since $T_\gamma \leq L_\gamma \leq U_\gamma$, we have $T_\kappa \leq L_\kappa \leq U_\kappa$. $\square$

Corollary 1. *If γ is a path to an always-passing leaf found by best-first search, and α is the result of unsetting all non-reducing inputs in γ, then α is a prime implicant of the decision tree's Boolean function.*

Proof. Since γ reaches an always-passing leaf, γ is an implicant. By Lemma 1, we can unset the non-reducing inputs in γ to get another implicant κ. We cannot unset a reducing input from κ. If we could, we could unset instead the input used to reach the leaf (since it closes the gap less), so the parent of the leaf should have been always-passing. Hence, κ must be prime. $\square$

It thus follows that a pruned decision tree represents a prime cover of f.

Theorem 2. *Say we have a threshold test f, and one of its pruned decision trees found by best-first search. Suppose we take all paths to always-passing leaves, and we unset all inputs that are non-reducing. The resulting set of terms represents a prime cover of the threshold test's Boolean function.*

By enumerating always-passing leaves, we enumerate prime implicants of a threshold test's Boolean function f. Similarly, by enumerating always-failing leaves, we enumerate the prime implicants of $\neg f$. Thus, we can produce tightening inner and outer bounds on f, as in Equation 4.

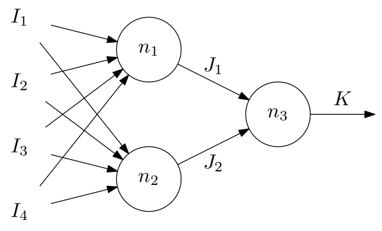
5 Bounding the Behavior of a Simple Neural Network

Consider a multilayer feedforward neural network. Using the best-first algorithm that we proposed in the previous section, we can bound the behavior of each neuron in the network, by enumerating its prime implicants. The more prime implicants that we enumerate from a neuron, the tighter the inner and outer bounds that we can obtain on it. In this section, we want to show how we can aggregate the bounds on the behavior of neurons, and propagate these bounds through the layers of the network, so that we obtain bounds on the behavior of the neural network itself.[2]

In this paper, we consider a simple class of neural networks, and show how we can obtain bounds on the network from bounds on its neurons. Consider, in particular, neural networks with:

- n inputs $I_1, \ldots, I_n$,
- two binary neurons n_1 and n_2 in a hidden layer,
- and a single binary output neuron n_3.

For example, the following neural network has 4 inputs I_1, I_2, I_3 and I_4:



This is perhaps the simplest type of architecture that surpasses the expressive ability of an individual neuron. For example, this type of structure can capture the behavior of an exclusive-or (XOR) function (over 2 inputs), which cannot be represented using a single neuron (or perceptron),[27,28] since XOR is not linearly-separable.

The inputs to neurons n_1 and n_2 are I_1, I_2, I_3 and I_4. The inputs to neuron n_3 are J_1 and J_2, which correspond to the outputs of neurons n_1 and n_2. The output of neuron n_3 is K. The neural network itself has inputs I_1, I_2, I_3 and I_4, whereas the output of the neural network is K.

In the previous section, we enumerated all input settings that cause a neuron to output a one, or equivalently, cause the neuron's threshold test to pass. Here,

[2]We distinguish related but orthogonal work based on propagating interval bounds on ReLU networks.[23–26] In such works, the network inputs and outputs are real-valued, and intervals on the values of the inputs are propagated through the layers of a neural network to obtain intervals on the value of the network output. Such an analysis can, for example, be used to verify the robustness of a class label, e.g., that small perturbations in the input will not significantly change the output.

we want to enumerate all input settings of a neural network that cause the *neural network* to output a one. In our simple network, to reason about what settings of the inputs I_1, I_2, I_3 and I_4 will cause the network output K to be a one, we need to

1. reason about what settings of the inputs J_1 and J_2 that will cause the output K of neuron n_3 to be a one, and in turn,
2. reason about what settings of inputs I_1, I_2, I_3 and I_4 that will (jointly) lead to the desired outputs J_1 and J_2 of neurons n_1 and n_2.

For example, if J_1 and J_2 both need to be one for output K of neuron n_3 to be one, we need to find settings of inputs I_1, I_2, I_3 and I_4 that cause the outputs J_1 and J_2 of neurons n_1 and n_2 to be one, at the same time.

5.1 Reasoning about the Behavior of the Output Neuron

Consider the output neuron n_3, which has two binary inputs, J_1 and J_2, with the corresponding threshold test f:

$$w_1 \cdot J_1 + w_2 \cdot J_2 \geq T.$$

As in Section 2, this threshold test has a corresponding truth table

J_1	J_2	f
0	0	a
0	1	b
1	0	c
1	1	d

where the output values a, b, c and d depend on the weights w_1 and w_2, as well as the threshold T. For example, if $w_1 = w_2 = 1$ and $T = \frac{1}{2}$, then we obtain (by enumeration) the truth table on the left. As another example, if $w_1 = w_2 = 1$ and $T = \frac{3}{2}$, then we obtain the truth table on the right.

J_1	J_2	f
0	0	0
0	1	1
1	0	1
1	1	1

J_1	J_2	f
0	0	0
0	1	0
1	0	0
1	1	1

The truth table on the left corresponds to a logical OR function. The one on the right corresponds to a logical AND function.

Depending on the weights w_1, w_2 and threshold T of a neuron, (almost) any truth table could be obtained. First, observe that for a truth table over two variables, each of its four rows can be either 0 or 1, and hence there are $2 \cdot 2 \cdot 2 \cdot 2 = 16$ different truth tables. Second, note that each truth table corresponds to a unique Boolean function over two variables. As a result, we can characterize the neuron n_3 by the Boolean function that it represents. Finally, we remark that only 14 out of these 16 different Boolean functions are realizable by a binary neuron.

Remark 2. *A binary neuron with two binary inputs and a binary output corresponds to one of 14 out of 16 different Boolean functions over two variables.*

In particular, it is well-known that a linear threshold test (or equivalently, a perceptron) cannot represent the exclusive-or (XOR) function,[27] nor can it represent the equivalence (EQ) function:

J_1	J_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

J_1	J_2	EQ
0	0	1
0	1	0
1	0	0
1	1	1

.

If n_3 is the output neuron, then knowing the Boolean function that it represents will tell us what the inputs J_1 and J_2 need to be for the neural network to output a one (in particular, it suffices to enumerate its truth table). Next, we need to reason about what the networks inputs $I_1, \ldots, I_n$ need to be so that the hidden neurons n_1 and n_2 will output the desired values of J_1 and J_2.

5.2 Reasoning about the Behavior of the Hidden Neurons

Suppose that the output neuron, n_3, corresponds to a logical AND function. In order for the output K to be one, then both of the inputs J_1 and J_2 need to be one. We next want to characterize when the inputs $I_1, \ldots, I_n$ of the neural network will lead J_1 and J_2 to be one.

When we consider the hidden neurons n_1 and n_2, some input settings will lead to the output J_1 of n_1 to be one, and some other input settings will lead to the output J_2 of n_2 to be one. Those input settings that are *common* will lead to both the outputs J_1 and J_2 to be one *at the same time.* These are precisely the input settings $I_1, \ldots, I_n$ that will cause the output neuron n_3 to be one, when it corresponds to a logical AND.

By enumerating the prime implicants of neurons n_1 and n_2, we can obtain inner and outer bounds on when their outputs will be one. We want to combine these bounds, and propagate them through neuron n_3 to obtain inner and outer bounds on when the output of the neural network will be one. As it turns out, when the output neuron n_3 corresponds to a logical AND, it suffices to conjoin the bounds of the hidden neurons, to obtain bounds on the neural network itself. Neurons corresponding to other logical functions call for other ways to aggregate their bounds, as we shall soon see.

Suppose that f is (the Boolean function of) a threshold test. Let f_i denote an inner bound of f, and let f_o denote an outer bound of f:

$$f_i \models f \models f_o.$$

Figure 2 (top) visualizes these inner and outer bounds. In particular, each box visualizes the relationship between a function f and one of its bounds, either the inner bound f_i (top-left) or the outer bound f_o (top-right). Each box itself represents all possible inputs to a function f. Some of those inputs satisfy f

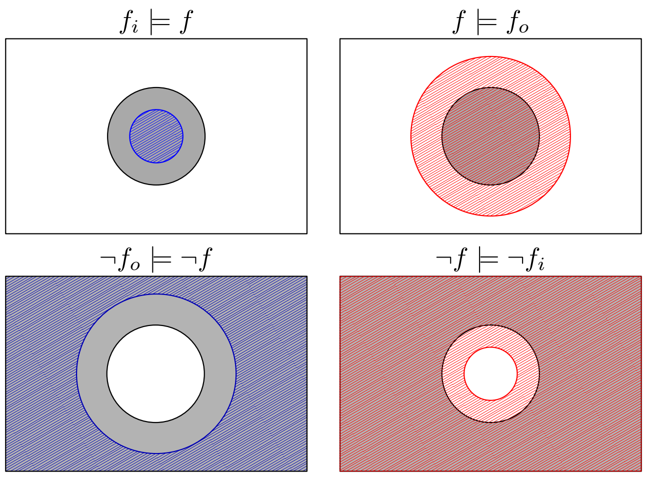


Figure 2: Inner and outer bounds for the original function f (top) and its negation $\neg f$ (bottom). The gray shaded region represents the function being bounded, the blue hatched region represents the inner bound, and the red hatched region represents the outer bound.

(gray shaded region). An inner bound f_i (blue hatched region) covers some, but not all of those satisfying inputs. An outer bound f_o (red hatched region) covers all of the satisfying inputs, but may cover some inputs that do not satisfy the original function.

Given the inner and outer bounds of a function f, or of a pair of functions g and h, we can obtain the inner and outer bounds of their negation, conjunction, and disjunction, as we show next.

Lemma 2. *If we have inner and outer bounds on a function $f_i \models f \models f_o$, then we have inner and outer bounds on the negation:*

$$\neg f_o \models \neg f \models \neg f_i.$$

If we have inner and outer bounds on a pair of functions, $g_i \models g \models g_o$ and $h_i \models h \models h_o$, then we have inner and outer bounds on their conjunction:

$$g_i \wedge h_i \models g \wedge h \models g_o \wedge h_o,$$

and we have inner and outer bounds on their disjunction:

$$g_i \vee h_i \models g \vee h \models g_o \vee h_o.$$

Proof. First, we review some basic definitions. Let ω denote a *world*: an assignment of every propositional variable to a value, i.e., a truth assignment. If α is a propositional sentence, then we say that ω *entails* α, denoted $\omega \models \alpha$, iff ω satisfies α. We say that a sentence α entails (or implies) a sentence β, denoted $\alpha \models \beta$, iff for all worlds ω, we have that $\omega \models \alpha$ implies $\omega \models \beta$.

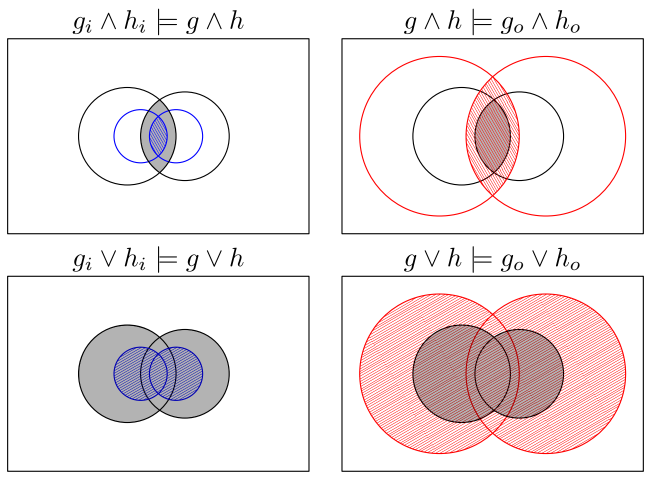

$g_i \wedge h_i \models g \wedge h$
$g \wedge h \models g_o \wedge h_o$
$g_i \vee h_i \models g \vee h$
$g \vee h \models g_o \vee h_o$

Figure 3: Inner and outer bounds for the conjunction $g \wedge h$ (top) and for the disjunction $g \vee h$ (bottom). The gray shaded region represents the function being bounded, the blue hatched region represents the inner bound, and the red hatched region represents the outer bound.

(Negation) We show that $f_i \models f$ implies $\neg f \models \neg f_i$; the same argument can be used to show that $f \models f_o$ implies $\neg f_o \models \neg f$. First, assume $f_i \models f$. By the definition of entailment (of two sentences), if $\omega \models f_i$ then $\omega \models f$. From the contrapositive, if $\omega \not\models f$ then $\omega \not\models f_i$. By the definition of negation, if $\omega \models \neg f$ then $\omega \models \neg f_i$. By the definition of entailment, $\neg f \models \neg f_i$.

(Conjunction) We show that $g_i \models g$ and $h_i \models h$ implies $g_i \wedge h_i \models g \wedge h$; the same argument can be used to show that $g \models g_o$ and $h \models h_o$ implies $g \wedge h \models g_o \wedge h_o$. First, assume $g_i \models g$ and $h_i \models h$. By the definition of entailment, if $\omega \models g_i$ then $\omega \models g$, and if $\omega \models h_i$ then $\omega \models h$. Thus, if $\omega \models g_i$ and $\omega \models h_i$, then $\omega \models g$ and $\omega \models h$. By the definition of conjunction, if $\omega \models g_i \wedge h_i$, then $\omega \models g \wedge h$. By the definition of entailment $g_i \wedge h_i \models g \wedge h$.

(Disjunction) The proof is analogous to the conjunction case. $\square$

Figures 2 & 3 visualize the bounds of Lemma 2. Consider first Figure 3. Again, each box visualizes the relationship between a function and one of its bounds, either the inner bound or the outer bound. The top row considers the conjunction $g \wedge h$ and the bottom row considers the disjunction $g \vee h$. The black circles represent the functions g and h, with the shaded region representing their conjunction (top) or their disjunction (bottom). The blue circles represent the inner bounds g_i and h_i, and the red circles represent the outer bounds g_o and h_o. The blue hatched regions represent the inner bounds of the conjunction and disjunction, and the red hatched regions represent the outer bounds.

Figure 2 visualizes the inner and outer bounds on the negation of the function $\neg f$. The gray shaded regions represent $\neg f$. The inner bound is found by taking the negation of the outer bound (the blue hatched region, which is a subset of

Table 1: Inner and Outer Bounds

id	$f(X,Y)$	truth table	inner bound	outer bound
0	$\perp$	0000	$\perp$	$\perp$
1	AND	0001	$g_i \wedge h_i$	$g_o \wedge h_o$
2	$\not\Rightarrow$	0010	$g_i \not\Rightarrow h_o$	$g_o \not\Rightarrow h_i$
3	X	0011	g_i	g_o
4	$\not\Leftarrow$	0100	$g_o \not\Leftarrow h_i$	$g_i \not\Leftarrow h_o$
5	Y	0101	h_i	h_o
6	XOR	0110	—	—
7	OR	0111	$g_i \vee h_i$	$g_o \vee h_o$
8	NOR	1000	$\neg(g_o \vee h_o)$	$\neg(g_i \vee h_i)$
9	EQ	1001	—	—
10	$\neg Y$	1010	$\neg h_o$	$\neg h_i$
11	$\Leftarrow$	1011	$g_i \Leftarrow h_o$	$g_o \Leftarrow h_i$
12	$\neg X$	1100	$\neg g_o$	$\neg g_i$
13	$\Rightarrow$	1101	$g_o \Rightarrow h_i$	$g_i \Rightarrow h_o$
14	NAND	1110	$\neg(g_o \wedge h_o)$	$\neg(g_i \wedge h_i)$
15	$\top$	1111	$\top$	$\top$

the gray shaded region). The outer bound is found by taking the negation of the inner bound (the red hatched region which is a superset of the gray shaded region).

Finally, Table 1 lists all 16 different Boolean functions $f(X,Y)$ over two variables X and Y. In the third column, the Boolean function's truth table is also provided in abbreviated form, i.e., *abcd* where a is the first row, b is the second row, etc. Table 1 also provides inner and outer bounds on the outputs of each Boolean function, given inner an outer bounds on its inputs.

Theorem 3. *Let $\circ$ denote a Boolean function over two variables. If $g_i \models g \models g_o$ and $h_i \models h \models h_o$, then Table 1 provides inner and outer bounds on $g \circ h$.*

Proof. Rows 0 (false) and 15 (true) are trivial. Row 3 (X) and Row 5 (Y) are given. Row 12 ($\neg X$) and Row 10 ($\neg Y$) follow from Lemma 2 (negation case).

Row 1 (AND) and Row 7 (OR) follow from Lemma 2 (by applying the conjunction and disjunction cases, respectively). Row 14 (NAND) and Row 8 (NOR) follow next by applying the negation case.

Since $X \Rightarrow Y \equiv \neg X \vee Y$, and $X \Leftarrow Y \equiv X \vee \neg Y$, Row 13 ($\Rightarrow$) and Row 11 ($\Leftarrow$) follow from Lemma 2 (by applying both the negation and disjunction cases). Row 2 ($\not\Rightarrow$) and Row 4 ($\not\Leftarrow$) follow next by applying the negation case again. $\square$

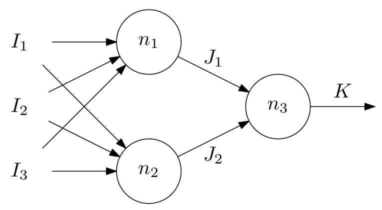
Theorem 3 thus tells us how to obtain inner and outer bounds of our simple neural network, given inner and outer bounds on the hidden neurons n_1 and n_2.

5.3 Towards Bounding the Behavior of Deep Neural Networks

We just showed how to bound the behavior of a simple class of neural network architectures. In particular, since we assume a single hidden layer of two neurons, the output neuron has two inputs, and hence, the behavior of the output neuron corresponds to one of the simple Boolean functions in Table 1. If the hidden layer has three or more neurons, the output neuron has three or more inputs. In this case, the Boolean function of the output neuron, may no longer be one of the simple ones from Table 1. In general, deep neural networks will also have many hidden layers, with each layer composed of many neurons. The behavior of the output neuron will in general correspond to a complex Boolean function. In principle, one can seek a circuit representation of this Boolean function.[17] One could then propagate bounds through the gates of the resulting Boolean circuit. We leave the investigation of the efficacy of this approach as future work.

6 A Complete Example

Consider the following simple neural network over three inputs I_1, I_2 and I_3:



Suppose the 3 neurons have the three corresponding threshold tests:

$$\begin{aligned} n_1: &\quad 3 \cdot I_1 + 2 \cdot I_2 - 4 \cdot I_3 \geq 1 \\ n_2: &\quad 4 \cdot I_1 - 1 \cdot I_2 - 3 \cdot I_3 \geq 1 \\ n_3: &\quad 2 \cdot J_1 + 2 \cdot J_2 \geq 3. \end{aligned}$$

Consider first the hidden neuron n_1. Let f be the Boolean function corresponding to neuron n_1, i.e., f represents all of the inputs where the threshold test for n_1 passes. Using the best-first search algorithm from the previous section, we can compute the prime cover $f = \alpha_1 \vee \alpha_2 \vee \alpha_3$ with three prime implicants:

$$\alpha_1 = I_1 \wedge \neg I_3 \qquad \alpha_2 = I_2 \wedge \neg I_3 \qquad \alpha_3 = I_1 \wedge I_2$$

Each prime implicant α_i summarizes a part of the input space where the threshold test passes, i.e., $\alpha_i \models f$. Further, any subset $\alpha_1 \vee \cdots \vee \alpha_k$ of the prime cover yields an inner bound f_i of the function f, i.e. $f_i \models f$. We can enumerate the prime implicants of f one-at-a-time yielding incrementally tighter inner bounds f_i:

I_1	I_2	I_3	f_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$f_i = \bot$

I_1	I_2	I_3	f_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	**0**	**0**	**1**
1	0	1	0
1	**1**	**0**	**1**
1	1	1	0

$f_i = \alpha_1$

I_1	I_2	I_3	f_i
0	0	0	0
0	0	1	0
0	**1**	**0**	**1**
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$f_i = \alpha_1 \vee \alpha_2$

I_1	I_2	I_3	f_i
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	**1**	**1**	**1**

$f_i = \alpha_1 \vee \alpha_2 \vee \alpha_3$

Let (I_1, I_2, I_3) denote an input vector. In the first table, when the prime cover is empty (we use no prime implicants), we obtain the trivial inner bound of false, denoted by $\bot$. The first prime implicant $\alpha_1 = I_1 \wedge \neg I_3$ covers two inputs $(1, 0, 0)$ and $(1, 1, 0)$, highlighted in bold. The second prime implicant $\alpha_2 = I_2 \wedge \neg I_3$ covers two input $(0, 1, 0)$ and $(1, 1, 0)$, where input $(0, 1, 0)$ was not already covered. This new input represents a tighter inner bound, and its addition is highlighted in bold. Finally, when we add the last prime implicant $\alpha_3 = I_1 \wedge I_2$, which corresponds to two inputs $(1, 1, 0)$ and $(1, 1, 1)$, we cover one additional input where the threshold test passes, i.e., $(1, 1, 1)$, highlighted in bold. At this point, we have covered all input cases when the threshold test passes, and our inner bound f_i now matches precisely the neuron's function f.

Let $\neg f$ be the Boolean function representing all of the inputs where the threshold test for n_1 fails. Using the best-first search algorithm from the previous section, we can compute the prime cover $\neg f = \beta_1 \vee \beta_2 \vee \beta_3$ with three prime implicants:

$$\beta_1 = \neg I_1 \wedge I_3 \qquad \beta_2 = \neg I_2 \wedge I_3 \qquad \beta_3 = \neg I_1 \wedge \neg I_2$$

Here, each prime implicant β_i summarizes a part of the input space where the threshold test *fails*, i.e., where $\beta_i \models \neg f$, and equivalently, $f \models \neg \beta_i$. Further, any subset $\neg \beta_1 \wedge \cdots \wedge \neg \beta_k$ of the prime cover yields an outer bound f_o of the function f, i.e. $f \models f_o$. We can enumerate the prime implicants of f one-at-a-time yielding incrementally tighter outer bounds f_o:

I_1	I_2	I_3	f_o
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$f_o = \top$

I_1	I_2	I_3	f_o
0	0	0	1
0	**0**	**1**	**0**
0	1	0	1
0	**1**	**1**	**0**
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$f_o = \neg \beta_1$

I_1	I_2	I_3	f_o
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	**0**	**1**	**0**
1	1	0	1
1	1	1	1

$f_o = \neg \beta_1 \wedge \neg \beta_2$

I_1	I_2	I_3	f_o
0	**0**	**0**	**0**
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$f_o = \neg \beta_1 \wedge \neg \beta_2 \wedge \neg \beta_3$

In the first table, when the prime cover is empty, we obtain the trivial outer bound of true, denoted by $\top$. The first prime implicant $\beta_1 = \neg I_1 \wedge I_3$ covers two inputs $(0, 0, 1)$ and $(0, 1, 1)$, highlighted in bold. The second prime implicant $\beta_2 = \neg I_2 \wedge I_3$ covers two input $(0, 0, 1)$ and $(1, 0, 1)$, where input $(1, 0, 1)$ was not already covered. This new input represents a tighter outer bound, and its addition is highlighted in bold. Finally, when we add the last prime implicant $\beta_3 = \neg I_1 \wedge \neg I_2$, which corresponds to two inputs $(0, 0, 0)$ and $(0, 0, 1)$, we cover one additional input where the threshold test fails, i.e., $(0, 0, 0)$, highlighted in bold. At this point, we have covered all input cases when the threshold test fails, and our outer bound f_o now matches precisely the neuron's function f (the same f that we arrived at when we fully tightened our inner bound).

Similarly, neuron n_2 has a Boolean function with a prime cover $g = \alpha_1 \vee \alpha_2$ for passing threshold tests, with two prime implicants:

$$\alpha_1 = I_1 \wedge \neg I_3 \qquad \alpha_2 = I_1 \wedge \neg I_2$$

and a prime cover $\neg g = \beta_1 \vee \beta_2$ for failing threshold tests, with two prime implicants:

$$\beta_1 = \neg I_1 \qquad \beta_2 = I_2 \wedge I_3$$

We have the following sequence of inner bounds:

I_1	I_2	I_3	g_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_i = \bot$

I_1	I_2	I_3	g_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	**0**	**0**	**1**
1	0	1	0
1	**1**	**0**	**1**
1	1	1	0

$g_i = \alpha_1$

I_1	I_2	I_3	g_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	**0**	**1**	**1**
1	1	0	1
1	1	1	0

$g_i = \alpha_1 \vee \alpha_2$

and the sequence of outer bounds:

I_1	I_2	I_3	g_o
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$g_o = \top$

I_1	I_2	I_3	g_o
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$g_o = \neg\beta_1$

I_1	I_2	I_3	g_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	**1**	**1**	**0**

$g_o = \neg\beta_1 \wedge \neg\beta_2$

Again, rows are in bold when they provide additional coverage of the input space.

Neuron n_3 is an output neuron with two binary inputs, and a binary output. Hence, it corresponds to one of 14 different Boolean functions over two variables, as in Remark 2. In this case, it corresponds to an AND function, via inspection. From Theorem 3 (and Lemma 2 more specifically), the bounds on an AND function can be obtained by conjoining the bounds on its inputs.

Consider the following table, representing the conjunction of the inner bounds.

$f_i \wedge g_i$	$g_i = 0000\|0000$	$g_i = 0000\|1010$	$g_i = 0000\|1110$
$f_i = 0000\|0000$	0000|0000	0000|0000	0000|0000
$f_i = 0000\|1010$	0000|0000	0000|1010	0000|1010
$f_i = 0010\|1010$	0000|0000	0000|1010	0000|1010
$f_i = 0010\|1011$	0000|0000	0000|1010	0000|1010

Each cell of the table corresponds to a truth table, using the abbreviation $abcd|efgh$, where a corresponds to the first row $(I_1, I_2, I_3) = (0, 0, 0)$ of the truth table, b corresponds to the second row $(I_1, I_2, I_3) = (0, 0, 1)$, etc. Each row of this table, from top-to-bottom, corresponds to increasingly tighter inner bounds g_i on neuron n_1. Each column, from left-to-right, corresponds to increasingly tighter inner bounds h_i on neuron n_2. Each cell represents a conjunction of two truth tables, one from neuron n_1 (from the row header) and the other from neuron n_2 (from the column header). In the upper-left corner, we have the trivial inner bound of false. As we go from top-to-bottom, and left-to-right, we discover more 1's of the truth table, indicating a tighter inner bound, until we obtain the exact function in the bottom-right corner.

We have a similar table for the outer bound on the output neuron n_3.

$f_o \wedge g_o$	$g_o = 1111\|1111$	$g_o = 0000\|1111$	$g_o = 0000\|1110$
$f_o = 1111\|1111$	1111|1111	0000|1111	0000|1110
$f_o = 1010\|1111$	1010|1111	0000|1111	0000|1110
$f_o = 1010\|1011$	1010|1011	0000|1011	0000|1010
$f_o = 0010\|1011$	0010|1011	0000|1011	0000|1010

As we go from top-to-bottom, and left-to-right, we discover more 0's of the truth table, indicating a tighter outer bound, until we obtain the exact function in the bottom-right corner.

7 Case Studies

We provide two case studies that highlight our ability to bound the behavior of a neuron. The first case study considers a dataset of handwritten digits, and the second case study considers a dataset of Congressional voting records.

7.1 Case Study: Handwritten Digits

We next show how to bound the behavior of a neuron using the MNIST dataset of handwritten digits. MNIST consists of 55,000 grayscale images, which we binarized to black-and-white. Each image has $28 \times 28 = 784$ pixels, and is labeled with a digit from 0 to 9. We consider one-vs.-one classification over all $\binom{10}{2} = 45$ pairs of digits (i, j). The resulting binary classifier for pair (i, j)

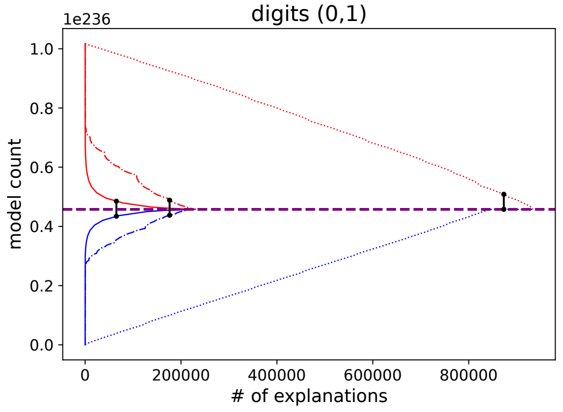


Figure 4: Bounding the behavior of a 0 versus 1 neuron.

corresponds to a Boolean function with 784 binary inputs and one binary output that is true or false if the digit is i or j. We used the `scikit-learn` Python library to train a binary neuron with a step activation.[3]

Consider Figure 4, which illustrates our ability to bound the behavior of a neuron, using the best-first search that we proposed. To visualize the inner and outer bounds of a neuron's Boolean function f, as in Equation 4, we plot lower and upper bounds on the *model count* of f, i.e., the number of its satisfying assignments. That is, the model count of the inner bound is a lower bound on the model count of f, and the model count of the outer bound is an upper bound on the model count.[4] As each implicant α that we enumerate yields a tighter inner and outer bound on f, they also yield a tighter lower and upper bound on the model count of f.

In Figure 4, where we considered 0-versus-1 digit classification, blue lines represent lower bounds and red lines represent upper bounds, which meet at the horizontal purple line representing the neuron's model count. The model count represents the number of input settings where the neuron outputs a 1 label, which is roughly 4.57×10^{235} out of $2^{784} \approx 1.01 \times 10^{236}$ possible input settings. Solid red & blue lines represent the best-first search (BFS) that we proposed. The dash-dotted lines represent a greedy depth-first search (DFS), where we set inputs with highest weight first, and to their reducing value first; greedy DFS represents the enumeration algorithm proposed by Marques-Silva

[3]We first trained a logistic regression model, with L_1 penalty, inverse regularization strength $C = 0.002$, and the `liblinear` solver, and then replaced the sigmoid with a step activation.

[4]Note that, if a Boolean function f has n inputs and an implicant α of f has k literals, then α represents 2^{n-k} models of f: there are $n-k$ missing inputs in α, and thus 2^{n-k} ways of completing α. By Theorem 2, we obtain a prime cover of a neuron's Boolean function by aggregating the *reduced* paths to the decision tree's leaves. However, the implicants of a prime cover are not mutually-exclusive (and may double-count models), whereas the *unreduced* paths are mutually-exclusive. Hence, to bound the model count, we use unreduced path costs in our best-first search.

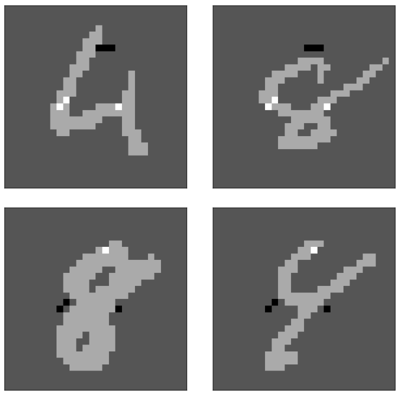


Figure 5: Digits classified as a 4 (top row) and as an 8 (bottom row), with their PI-explanations.

et al.[19] which is what our BFS is inspired by, and is what we seek to improve on. In BFS and greedy DFS, one search was performed to find the lower bound (the always-passing leaves) and a separate search was performed to find the upper bound (the always-failing leaves), as the reducing values for upper and lower bounds are different. The dotted lines represent a naive DFS where inputs were set in their natural order.

First, we observe that our BFS clearly obtains tighter lower and upper bounds compared to the alternatives. Naive DFS generates very loose bounds and nearly the entire decision tree needs to be enumerated before one obtains tight bounds. Greedy DFS performs much better than naive DFS, but not as well as our BFS. While greedy DFS explores more promising branches of the search tree first, once it goes down a branch, it must finish exploring it, unlike BFS which can explore more promising branches elsewhere. On the other hand, the space complexity of BFS is linear in the size of the search frontier, which may become exponentially large.

For each type of search, we also plotted using black vertical lines the point where each search enumerated 95% of the input space (or a 5% gap between the bounds), relative to the 2^{784} possible input settings. To reach this point, BFS enumerated only 65,176 implicants compared to 176,219 implicants enumerated by greedy DFS. Both searches enumerated 220,640 total implicants for the lower bound and 229,964 total implicants for the upper bound. In contrast, naive DFS had to enumerate 873,228 implicants out of 839,075 and 933,899 total implicants for its lower and upper bounds. For digit pair $(2, 8)$, greedy DFS enumerated 8.12 times more implicants than our BFS, and naive DFS enumerated 418.52 times more than our BFS. On average, across all 45 pairs of digits, greedy DFS enumerated 2.74 times more implicants, and naive DFS enumerated 51.11 times more.

Finally, we highlight how prime implicants can be used to gain insights on the behavior of a classifier. Figure 5 highlights the results of our 4-vs.-8 digit

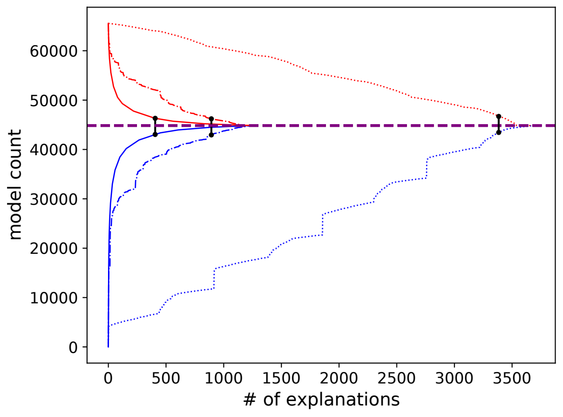


Figure 6: Bounding the behavior of neuron for Congressional voting.

classifier, which labeled the top two images as a 4, and the bottom two images as an 8. Consider the top-left image of a 4, where we have highlighted six pixels: three white pixels on the left and right of the image, and three black pixels near the top of the image. These six pixels are a prime-implicant (PI) explanation for why the classifier labeled this image as a 4.[7,9,20,21] In particular, these six pixels are sufficient for the classifier to label the image as a 4; none of the other pixels matter once these six pixels are fixed. Consider the bottom-left image of an 8, where the PI-explanation is composed of two black pixels on the left and right and one white pixel near the top. These two explanations suggest together that the classifier is using a simple pattern to label an image: looking at pixels on the left and right suggests that the classifier is trying to detect the presence of a horizontal stroke in a 4, or its absence in an 8. Similarly, looking at pixels on the top suggests that the classifier is trying to detect a closed loop for an 8, or an open one for a 4. This strategy was enough for the classifier to achieve a training set accuracy of 92.88%, but we are clearly not learning a robust representation of 4's and 8's. The right column of Figure 5 shows images of an 8 and 4 that evaded this simple pattern, and were thus misclassified.

7.2 Case Study: Congressional Voting

Next, we present a case study using the 1984 Congressional voting records dataset, from the UCI ML Repository.[29] This dataset consists of 435 examples, one for each member of the U.S. House of Representatives, with 16 attributes, corresponding to 16 key votes. Each instance is labeled as Republican (R) or Democrat (D); the classification task is to predict a member's party given their key votes. Each key vote can be a `y` for a yea vote, an `n` for a nay vote, and a `?` if the vote was missing.

After imputing missing votes, based on the majority vote of the member's party, the resulting dataset is binary. A 1 output (passing threshold test) corresponds to a Democrat label, and a 0 output (failing threshold test) corresponds to a Republican label. We used the `scikit-learn` library to train a binary neuron,[5] with a training accuracy of 97.70%.

Figure 6 highlights the ability of our enumeration algorithm to bound the behavior of the neuron that we trained from this dataset. To cover the total input space, our BFS enumerated 1,243 implicants for the lower bound (blue solid line) and 1,193 implicants for the upper bound (red solid line). In order to cover 95% of the input space (highlighted with black vertical bars), our BFS (solid lines) needed to enumerate only 406 implicants, whereas greedy DFS (dashed-dotted lines) needed to enumerate over twice as many, using 893 implicants. Naive DFS (dotted lines) needed to enumerate 3,384 implicants.

Table 2 highlights three of the shortest PI-explanations for Republicans (R) and Democrats (D). We see that, for the neuron that we trained, as few as 5 (of the right) votes are needed to label a Congressmember as a Republican, and as few as 3 are needed to label a Congressmember as a Democrat. A PI-explanation's votes are sufficient to determine the behavior of the classifier: the values of the remaining votes would not change the classifier's decision (i.e., label).

Table 2 also highlights the vote counts by party for each key vote. From the PI-explanations, Bills 3 and 4 appear to be important in determining whether one is a Republican or a Democrat. From the vote counts, Bill 3 was heavily favored by Democrats and heavily opposed by Republicans, and vice-versa for Bill 4.[6] These two bills were not sufficient to commit to a label; in addition, some combination of votes on Bills 9, 10, 11 and 12 were also used by the classifier.[7]

Finally, consider Figure 7 where we highlight our ability to bound the behavior of a simple neural network trained from the same dataset. As in Section 5, we trained a binary neural network with two hidden neurons and one output neuron, using `tensorflow`,[8] where we obtained a neural network with 98.39% accuracy. Using the best-first search (BFS) algorithm we proposed in Section 3 (which was more efficient than greedy and naive DFS) we enumerated a prime cover for both hidden neurons, where neuron n_1 had 965 prime implicants, and neuron n_2 had 1,070 prime implicants. We further found that the output neuron corresponded to a logical OR function, by inspection. In Figure 7, we visualize the inner bound on the behavior of the neural network, by plotting the lower bound on the model count (the model count was $48,638$). As in Theorem 3, the inner bound on the network is found by disjoining the inner bounds of the hidden neurons. On the x and y axes, we increase the number of prime impli-

[5]We trained a logistic regression model with default parameters.

[6]Bill 3 proposed to raise taxes, lower military spending, and raise domestic spending. Bill 4 proposed a one-year freeze on physicians' fees, in an effort to help curb rising healthcare costs.

[7]Bill 9 proposed to regulate funding on intercontinental missiles, Bill 10 proposed to restrict the hiring of unauthorized workers, Bill 11 proposed to decrease funding for synthetic fuels, and Bill 12 proposed an income tax deduction for educational expenses.

[8]We assumed sigmoid activations, and minimized mean-squared-error using the Adam optimizer.

Table 2: 1984 Congressional Voting Records

bills		PI-explanations						vote counts					
		R_1	R_2	R_3	D_1	D_2	D_3	R_Y	R_N	$R_?$	D_Y	D_N	$D_?$
1	handicapped-infants	-	-	-	-	-	-	31	134	3	156	102	9
2	water-project-cost-sharing	-	-	-	-	-	-	75	73	20	120	119	28
3	budget-resolution	n	n	n	y	y	y	22	142	4	231	29	7
4	physician-fee-freeze	y	y	y	n	n	n	163	2	3	14	245	8
5	el-salvador-aid	-	-	-	-	-	-	157	8	3	55	200	12
6	religious-groups-in-schools	-	-	-	-	-	-	149	17	2	123	135	9
7	anti-satellite-test-ban	-	-	-	-	-	-	39	123	6	200	59	8
8	aid-to-nicaraguan-contras	-	-	-	-	-	-	24	133	11	218	45	4
9	mx-missile	-	n	n	-	-	-	19	146	3	188	60	19
10	immigration	y	y	-	n	-	-	92	73	3	124	139	4
11	synfuels-corp-cutback	n	n	n	-	-	y	21	138	9	129	126	12
12	education-spending	y	-	y	-	n	-	135	20	13	36	213	18
13	superfund-right-to-sue	-	-	-	-	-	-	136	22	10	73	179	15
14	crime	-	-	-	-	-	-	158	3	7	90	167	10
15	duty-free-exports	-	-	-	-	-	-	14	142	12	160	91	16
16	export-admin-south-africa	-	-	-	-	-	-	96	50	22	173	12	82

cants (explanations) used to form the inner bound on each neuron. When we use our BFS algorithm to tighten the lower bounds of each input neuron, we find that we are able to combine the resulting bounds to obtain rapidly tightening lower bounds on the model count of our simple neural network. That is, relatively few prime implicants need to be enumerated from each neuron until the lower bound plateaus (less than 200 each). Moreover, in this case, we also find that it is more important to enumerate prime implicants from neuron n_1, as it contributes more to the final model count of the neural network.

8 Related Work

In the domain of explainable AI (XAI), prime implicants have been used extensively to explain the behavior of machine learning classifiers, where they are sometimes referred to as PI-explanations, or sufficient explanations.[7,9,20,21] Anchors are another popular approach to explanations; an Anchor can also be viewed as a probabilistic version of a prime implicant.[3,30] For example, a PI-explanation for why an image classifier labels an image a dog, would find a (small) sub-image that is sufficient for the classifier to commit to this decision. That is, it would find the smallest sub-image that is sufficient for the classifier to label the image as a dog. These types of explanations are sometimes referred to as a *local* explanation of a classifier's behavior (around a specific input). Prime implicants, viewed as a decomposition of a classifier's Boolean function are sometimes referred to as a *global* explanation of a classifier's behavior. In the latter case, one generally wants the ability to enumerate a classifier's prime implicants.

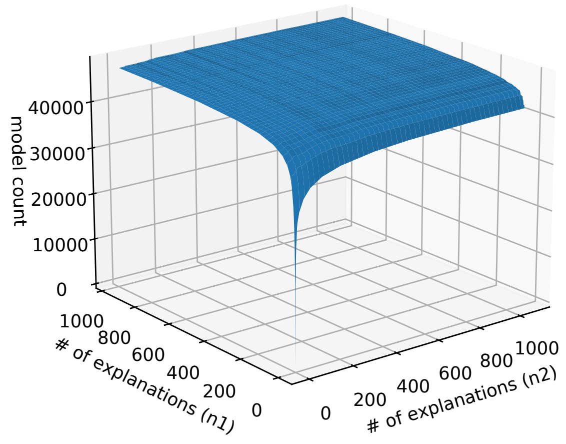


Figure 7: Bounding the behavior of a simple neural network for Congressional voting.

Shih et al. proposed an approach for enumerating prime implicants for Bayesian network classifiers, like naive Bayes classifiers.[7] Their algorithm could also enumerate prime implicants by size, from shortest-to-longest. However, their approach is based on compiling a naive Bayes classifier to OBDD, which is NP-hard.[7] Further, their enumeration algorithm[31] does not have (known) polynomial runtime guarantees. Marques-Silva et al. showed that prime implicants can in fact be enumerated efficiently for linear classifiers (including naive Bayes classifiers), and with only a polynomial delay, i.e., only a polynomial amount of time is spent to enumerate each additional prime implicant.[19] While the first prime implicant is guaranteed to be the shortest, their enumeration algorithm does not guarantee that prime implicants will be enumerated in order by size. In contrast, the best-first search algorithm that we proposed in Section 4 will enumerate prime implicants from shortest-to-longest. However, we lose the guarantee of polynomial delay, although in Section 7, we found best-first search to be much more efficient in terms of covering the input space.

Bounds on the behavior of a neural network have also been developed for verifying the robustness of neural networks.[25, 26] Given an input vector to a neural network, the goal here is to provide a guarantee that small perturbations to the input will not cause a big change in the output. For example, we may specify bounds on each input, and seek to propagate such bounds from one layer to the next (while possibly weakening the bounds). Such approaches are similar in spirit to the one we proposed in Section 5, although are goals differ. Our approach seeks to bound the global behavior of a network (i.e., bound the sub-space of the input that leads to a positive output) versus verifying bounds on the local behavior of a network (i.e., bound the range of the output based on small perturbations to the input).

9 Conclusion

We proposed an approach that incrementally tightens inner and outer bounds on the behavior of a binary neuron. We build on a recently proposed approach for efficiently enumerating prime implicants from a linear classifier. We simplify the problem of enumerating prime implicants, and formulate it as a best-first search in a space over threshold tests. We show that the inner and outer bounds correspond to a truncated prime implicant cover of a neuron's Boolean function. We further show how these bounds can be propagated through a simple class of neural networks. Through two case studies, we showed how our best-first search approach can quickly provide near-total coverage on the behavior of neurons, compared to other approaches.

Acknowledgments

The authors were supported by the Office of Undergraduate Research at Kennesaw State University. We thank Lance Kennedy for valuable feedback.

References

[1] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen and K. Müller, How to explain individual classification decisions, *Journal of Machine Learning Research (JMLR)* **11** (2010) 1803–1831.

[2] M. T. Ribeiro, S. Singh and C. Guestrin, "Why should I trust you?": Explaining the predictions of any classifier, in *Knowledge Discovery and Data Mining (KDD)* 2016.

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

[4] Z. C. Lipton, The mythos of model interpretability, *Communications of the ACM (CACM)* **61**(10) (2018) 36–43.

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

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

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

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

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

[10] G. Audemard, F. Koriche and P. Marquis, On tractable XAI queries based on compiled representations, in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR)* 2020.

[11] M. C. Cooper and J. Marques-Silva, On the tractability of explaining decisions of classifiers, in *27th International Conference on Principles and Practice of Constraint Programming (CP)* **210**, 2021, pp. 21:1–21:18.

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

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

[14] I. Wegener, *Branching Programs and Binary Decision Diagrams* (SIAM, 2000).

[15] A. Darwiche and P. Marquis, A knowledge compilation map, *Journal of Artificial Intelligence Research (JAIR)* **17** (2002) 229–264.

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

[17] W. Shi, A. Shih, A. Darwiche and A. Choi, On tractable representations of binary neural networks, in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR)* 2020.

[18] L. Kennedy, I. Kindo and A. Choi, On training neurons with bounded compilations, in *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR)* 2023.

[19] J. Marques-Silva, T. Gerspacher, M. C. Cooper, A. Ignatiev and N. Narodytska, Explaining naive Bayes and other linear classifiers with polynomial time and delay, in *Advances in Neural Information Processing Systems 33 (NeurIPS)* 2020.

[20] A. Ignatiev, N. Narodytska and J. Marques-Silva, Abduction-based explanations for machine learning models, in *Proceedings of The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)* 2019, pp. 1511–1519.

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

[22] O. Coudert, J. C. Madre, H. Fraisse and H. Touati, Implicit prime cover computation: An overview, in *Proceedings of the 4th SASIMI Workshop* 1993.

[23] E. Wong and J. Z. Kolter, Provable defenses against adversarial examples via the convex outer adversarial polytope, in *Proceedings of the 35th International Conference on Machine Learning (ICML)* 2018, pp. 5283–5292.

[24] H. Zhang, T. Weng, P. Chen, C. Hsieh and L. Daniel, Efficient neural network robustness certification with general activation functions, in *Advances in Neural Information Processing Systems 31 (NeurIPS)* 2018, pp. 4944–4953.

[25] A. Albarghouthi, Introduction to neural network verification, *Found. Trends Program. Lang.* **7**(1-2) (2021) 1–157.

[26] P.-Y. Chen and C.-J. Hsieh, *Adversarial Robustness for Machine Learning* (Academic Press, 2022).

[27] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry* (MIT Press, 1969).

[28] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning* (MIT Press, 2016).

[29] K. Bache and M. Lichman, UCI machine learning repository (2013).

[30] A. Ignatiev, N. Narodytska and J. Marques-Silva, On validating, repairing and refining heuristic ML explanations, *CoRR* **abs/1907.02509** (2019).

[31] O. Coudert and J. C. Madre, Fault tree analysis: 10^{20} prime implicants and beyond, in *Proc. of the Annual Reliability and Maintainability Symposium* 1993, pp. 240–245.