On Bounding the Behavior of a Neuron

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

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 propose a best-first search algorithm that is able to incrementally tighten inner and outer bounds of a neuron's Boolean function. These bounds correspond to truncated prime-implicant covers of the Boolean function. We provide two case studies that highlight our ability to bound the behavior of a neuron.

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); see, e.g. (Baehrens et al. 2010; Ribeiro, Singh, and Guestrin 2016; 2018; Lipton 2018). *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); see, e.g., (Katz et al. 2017; Leofante et al. 2018; Shih, Choi, and Darwiche 2018b; 2018a; Ignatiev, Narodytska, and Marques-Silva 2019b; Audemard, Koriche, and Marquis 2020; Cooper and Marques-Silva 2021).

Unfortunately, for a sufficiently powerful notion of explanation, it is NP-hard to explain the behavior of a neural network (Cooper and Marques-Silva 2021). For example, deciding if a neural network ever produces a positive labeling is analogous to testing the satisfiability of a Boolean formula. The situation is worse than this: 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 a 0. It is also NP-hard to compile an individual neuron into a more tractable representation, such as an Ordered Binary Decision Diagram (OBDD) (Shih, Choi, and Darwiche 2018b).[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 (Chan and Darwiche 2003; Shi et al. 2020). More recently, (Marques-Silva et al. 2020) showed that prime implicants (PIs) can be efficiently enumerated from a linear classifier. Prime implicants, and the corresponding PI-explanations (or sufficient explanations) provide a partial description of the behavior of a classifier (Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019a; 2019b; Darwiche and Hirth 2020). 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.

By enumerating the prime implicants of the Boolean function of a neuron, we obtain an inner and outer bound on the behavior of a neuron. In this paper, we formulate this enumeration problem as a best-first search problem, in order to obtain the tightest possible bounds. This yields a polytime approximation of a neuron's Boolean function, corresponding to a truncated prime implicant cover. Empirically, through two case studies, we show how our algorithm is able to provide near-total coverage of a neuron's behavior, by enumerating a relatively small number of prime implicants.

This paper is organized as follows. First, we characterize the behavior of a neuron in simpler terms, as a threshold test. We next show how the behavior of a threshold test can be bounded by prime implicants. Next, 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. Before concluding, we provide two case studies that highlight our ability to bound the behavior of a neuron.

[1] An OBDD is a *tractable* representation of a Boolean function that supports polynomial time transformations and operations (Bryant 1986; Meinel and Theobald 1998; Wegener 2000), which facilitate the explanation and formal verification of a neuron (Shih, Choi, and Darwiche 2018a; Audemard, Koriche, and Marquis 2020). 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; see, e.g., (Darwiche and Marquis 2002).

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.

Observation 1. *The output of a 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 output, as we would 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:*

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

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.

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 (Coudert et al. 1993). 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.[2] These implicants in aggregate provide

[2]Prime implicants have been applied towards explaining the behavior of machine learning classifiers previously, where they are referred to as PI-explanations, or sufficient explanations (Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019a; 2019b; Darwiche and Hirth 2020); probabilistic variations have also been proposed, such as Anchors (Ribeiro, Singh, and Guestrin 2018; Ignatiev, Narodytska, and Marques-Silva 2019c). 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 this decision, i.e., a (small) subset of pixels that need to be set before the classifier commits to its label. 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.

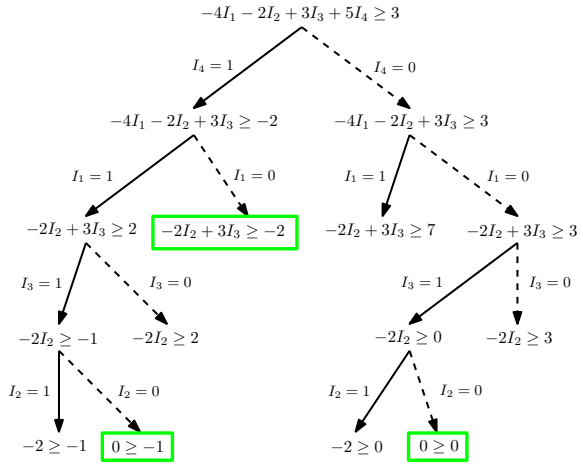

Figure 1: Decision tree over threshold tests.

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. 2020) 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.*

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

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:

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

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, and we have 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:

- $w \geq 0$ and $I = 0$: range $[L, U - w]$ and threshold T
- $w \geq 0$ and $I = 1$: range $[L, U - w]$ and threshold $T - w$
- $w < 0$ and $I = 0$: range $[L - w, U]$ and threshold T
- $w < 0$ and $I = 1$: range $[L - w, U]$ and threshold $T - w$

We 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 $L < T$ and the gap before it becomes always passing is $T - L$.*
- *If the threshold test is not always failing, then $T \leq U$ and the gap before it 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. 2020) 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.*

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).

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. 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 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 (Marques-Silva et al. 2020). 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).

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 of f.

If the decision tree has been pruned, 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.

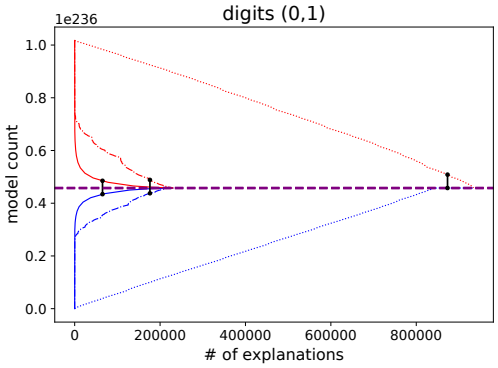

Figure 2: Bounding the behavior of a 0 versus 1 neuron.

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.*

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.*

See Appendix for proofs. From Lemma 1 and Corollary 1, 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.

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) 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` to train a binary neuron with a step activation.[3]

Consider Figure 2, which illustrates our ability to bound the behavior of a neuron, using the best-first search that we

[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.

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 2, 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 et al. 2020). 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.

[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.

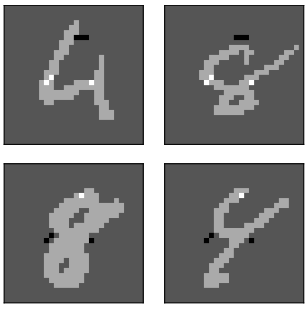

Figure 3: Digits classified as a 4 (top row), and classified as an 8 (bottom row), with their PI-explanations.

Finally, we highlight how prime implicants can be used to gain insights on the behavior of a classifier. Figure 3 highlights the results of our 4-vs.-8 digit 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 (Shih, Choi, and Darwiche 2018b; Ignatiev, Narodytska, and Marques-Silva 2019a; 2019b; Darwiche and Hirth 2020). 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 3 shows images of an 8 and 4 that evaded this simple pattern, and were thus misclassified.

Case Study: Congressional Voting

Next, we present a case study using the 1984 Congressional voting records dataset, from the UCI ML Repository (Bache and Lichman 2013). 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), and hence 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 imputating 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

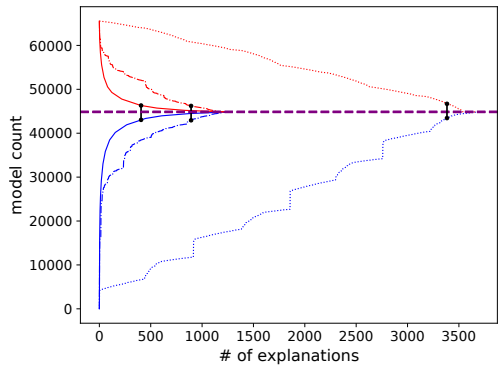

Figure 4: Bounding the behavior of a Congress vote neuron.

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 4 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 1 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 1 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]

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. Through two case studies, we showed how our best-first search approach can quickly provide near-total coverage on the behavior of a neuron, compared to other approaches.

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

Table 1: 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 infants	-	-	-	-	-	-	31	134	3	156	102	9
2 water-proj	-	-	-	-	-	-	75	73	20	120	119	28
3 budget-res	n	n	n	y	y	y	22	142	4	231	29	7
4 physicians	y	y	y	n	n	n	163	2	3	14	245	8
5 el-salvador	-	-	-	-	-	-	157	8	3	55	200	12
6 schools	-	-	-	-	-	-	149	17	2	123	135	9
7 anti-satellite	-	-	-	-	-	-	39	123	6	200	59	8
8 contra-aid	-	-	-	-	-	-	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	n	n	n	-	-	y	21	138	9	129	126	12
12 education	y	-	y	-	n	-	135	20	13	36	213	18
13 superfund	-	-	-	-	-	-	136	22	10	73	179	15
14 crime	-	-	-	-	-	-	158	3	7	90	167	10
15 duty-free	-	-	-	-	-	-	14	142	12	160	91	16
16 south-africa	-	-	-	-	-	-	96	50	22	173	12	82

Proofs

Proof of Lemma 1. 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 $T_\kappa \leq L_\kappa \leq U_\kappa$. □

Proof of Corollary 1. 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. □

[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.

References

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

Bache, K., and Lichman, M. 2013. UCI machine learning repository.

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.

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

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.

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

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

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

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

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

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

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019c. On validating, repairing and refining heuristic ML explanations. *CoRR* abs/1907.02509.

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.

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

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

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)*.

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

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)*.

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