

Specialized Neural Network Pruning for Boolean Abstractions

Jarren Briscoe^{1,2}^a, Brian Rague¹^b, Kyle Feuz¹^c and Robert Ball¹^d

¹Department of Computer Science, Weber State University, Ogden, Utah, U.S.A.

²Department of Computer Science, Washington State University, Pullman, Washington, U.S.A.

Keywords: Neural Networks, Network Pruning, Boolean Abstraction, Explainable AI, XAI, Interpretability.

Abstract: The inherent intricate topology of a neural network (NN) decreases our understanding of its function and purpose. Neural network abstraction and analysis techniques are designed to increase the comprehensibility of these computing structures. To achieve a more concise and interpretable representation of a NN as a Boolean graph (BG), we introduce the Neural Constantness Heuristic (NCH), Neural Constant Propagation (NCP), shared logic, the Neural Real-Valued Constantness Heuristic (NRVCH), and negligible neural nodes. These techniques reduce a neural layer's input space and the number of nodes for a problem in NP (reducing its complexity). Additionally, we contrast two parsing methods that translate NNs to BGs: reverse traversal (\mathcal{N}) and forward traversal (\mathcal{F}). For most use cases, the combination of NRVCH, NCP, and \mathcal{N} is the best choice.

1 INTRODUCTION

1.1 Background

While NNs are a powerful machine learning tool, they are semantically labyrinthine. The most essential attribute lacking in the interpretability of NNs is conciseness (Briscoe, 2021). This explains why many authors have opted to use Boolean graphs (BGs) as an explanatory medium (Briscoe, 2021; Brudermueller et al., 2020; Shi et al., 2020; Choi et al., 2017; Chan and Darwiche, 2012). However, this problem is in NP, and we introduce heuristics and traversal methods to reduce the complexity.

Neural Networks. Neural networks train on a data set containing inputs and classifications (supervised learning), unlabelled data (unsupervised learning), or policies (reinforcement learning). After training, the neural network predicts classifications for new inputs. Since neural network structures and algorithms are quite diverse, in this paper we limit the scope of neural networks to the most common type. Specifically, they are acyclic, symmetric, first-order neural networks

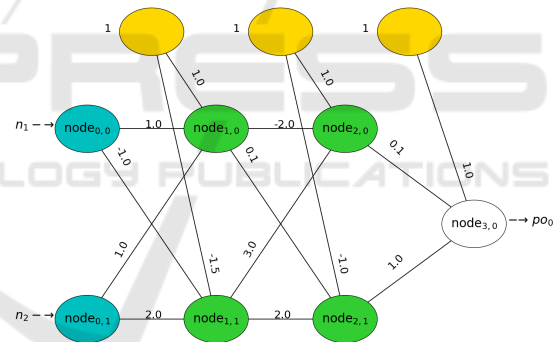





Figure 1: A multilayer perceptron. The hidden nodes (node_{1,0}, node_{1,1}, node_{2,0}, and node_{2,1}) and the output node (node_{3,0}) use the σ activation function. The legend's remainder is as follows: the input nodes are in the leftmost layer (node_{0,0} and node_{0,1}) and the bias nodes are in the top row (unlabeled).


with a finite amount of layers and nodes with binary input and real-valued activations (e.g., a multilayer perceptron such as Figure 1) or an autoencoder). For formal definitions of neural networks, see *Neural Network Formalization* (Fiesler, 1992).

Explaining Neural Networks. From the many techniques attempting to conceptualize and explain neural networks (Guidotti et al., 2018; Andrews et al., 1995; Baehrens et al., 2010; Brudermueller et al., 2020; Shi et al., 2020; Choi et al., 2017; Briscoe, 2021), three generic categories emerge: decomposi-

^a  <https://orcid.org/0000-0002-7422-9575>

^b  <https://orcid.org/0000-0001-9065-6780>

^c  <https://orcid.org/0000-0003-3730-3198>

^d  <https://orcid.org/0000-0001-5302-5293>

are several far more efficient methods. The exponentially upper-bound methods described below yield higher accuracy than the pseudo-polynomial methods that follow. We denote a generic NN to BG algorithm as \mathcal{B} .

Exponential Methods. An exponential method targeting Bayesian network classifiers reports a complexity of $O(2^{0.5n})$ (Chan and Darwiche, 2012). While this approximation was done for Bayesian network classifiers, the correlation to neural node approximation can be formalized (Choi et al., 2017). Another algorithm to approximate neural nodes exhibits a complexity of $O(2^n)$ (Briscoe, 2021).

Current Work. \mathcal{N} improves the upper-bound exponential complexity of Chan and Darwiche’s algorithm as applied to neural networks (Choi et al., 2017) with no loss in accuracy. Assuming a ratio $x : T$ of constant nodes to the total number of nodes, the new complexity becomes $O(2^{0.5n(1-x)})$.

Pseudo-Polynomial Methods Faster methods that compute a neural node with less accuracy produce a pseudo-polynomial $O(n^2W)^1$ complexity where

$$W = |\theta_D| + \sum_{w \in \text{node}_{i,j}} |w|. \quad (1)$$

θ_D is the threshold in the activation function’s domain, and $\text{node}_{i,j}$ is a vector of in-degree weights. Furthermore, these weights must be integers with fixed precision (Shi et al., 2020).

If one uses a constant maximum depth d for the binary decision tree (Briscoe, 2021), then the *entire* neural network can be computed in $O(n)$ time where n is the number of neural nodes. This is a pseudo-polynomial complexity since the constant d causes the individual neural node approximation to be a constant of $O(2^d) = O(1)$ and is still $O(2^n)$ for $n < d$. Of course, if d is not constant, then the neural node complexity becomes $O(2^{\min(n,d)})$.

We are currently working on creating a flexible maximum depth that considers the weight vector’s distribution and size. Future work could extend this idea to Chan and Darwiche’s algorithm on Bayesian Network classifiers and reduce the $O(2^{0.5n})$ complexity with minimal accuracy loss (and as employed by (Choi et al., 2017)). Enforcing a maximum depth is equivalent to capping a maximum amount

¹While the original text only mentions the $O(nW)$ complexity to compile the OBDD (Shi et al., 2020), the complexity to approximate the node is $O(n^2W)$.

of weights considered, and since the weights’ importance are ordered by the greatest absolute value (Briscoe, 2021), the weights with the smallest absolute values should be omitted first.

Final Product. The NN can be mapped to any Boolean structure. Shi et al. and Choi et al. used Ordered Binary Decision Diagrams (OBDDs) and Sentential Decision Diagrams (SDDs) while Brudermueller et al. and Briscoe employed And-Inverter Graphs (AIGs) (Shi et al., 2020; Choi et al., 2017; Briscoe, 2021; Brudermueller et al., 2020). However, other Boolean structures exist with different topological and/or operational features that highlight specific design (e.g. comprehensibility, Boolean optimizations, and hardware optimizations). The final product for this work is abstracted as a BG (Boolean Graph) and illustrated with ABC (Brayton and Mishchenko, 2010).

Neural Pruning. While traditional neural pruning methods seek to reduce overfitting, speedup neural networks, or even improve accuracy (Han et al., 2015), our pruning algorithm is specialized for interpretability and to reduce the worst-case complexity of \mathcal{B} . So our network pruning is not directly comparable to traditional pruning to the best of our knowledge.

4 ACTIVATION FUNCTIONS

Shi and Choi only considered ReLU and sigmoid functions (Shi et al., 2020; Choi et al., 2017). We extend the allowed activation functions to any that allow a reasonable Boolean cast (Conjunctive Limit Constraint) and conversion to a binary step function (Diagonal Quadrants Property).

Conjunctive Limit Constraint.

$$\lim_{\theta \rightarrow -\infty} f(\theta) = 0 \text{ and } \lim_{\theta \rightarrow \infty} f(\theta) > 0 \quad (2)$$

The conjunctive limit constraint allows us to treat one side of the range threshold as false (analogous to zero) and the other side as true (analogous to a positive number). Functions that have negative and positive limits (such as tanh) are better defined within an inhibitory/excitatory context. It can be experimentally shown that the accuracy for NN to BG methods decreases for functions like tanh. The Swiss activation function (Ramachandran et al., 2017) lies in between the inhibitory/excitatory and true/false spectrums since there are negative range values but it still satisfies the Conjunctive Limit Constraint.

Diagonal Quadrants Property. Converting a real-valued function ($f_{\mathbb{R}}$) to a binary-step function ($f_{\mathbb{B}}$) is best suited for real-valued functions that only lie in two diagonal quadrants or on the boundaries, centered on the threshold $f_{\mathbb{R}}(\theta_{\mathcal{D}}) = \theta_{\mathcal{R}}$. Consider the sigmoid function $\sigma_{\mathbb{R}}(\theta) = \frac{1}{1+e^{-\theta}}$. Since $\sigma_{\mathbb{R}}$'s range (\mathcal{R}) is in $(0, 1)$, we want the range's threshold, $\theta_{\mathcal{R}}$, to be 0.5 which corresponds to $\theta_{\mathcal{D}} = 0$ on the domain: $\sigma_{\mathbb{R}}(0) = 0.5$. Drawing horizontal and vertical lines through $\sigma_{\mathbb{R}}(0) = 0.5$ creates the quadrants. Since $\sigma_{\mathbb{R}}$ with threshold $\sigma_{\mathbb{R}}(0) = 0.5$ is contained in diagonal quadrants (top-right and bottom-left—see Figure 3), this activation function is suitable for conversion to a binary-step function $\sigma_{\mathbb{B}}$. With the threshold $\theta_{\mathcal{R}}$:

$$\sigma_{\mathbb{B}}(\theta; \sigma_{\mathbb{R}}, \theta_{\mathcal{R}}) = \begin{cases} 1 & \text{if } \sigma_{\mathbb{R}}(\theta) \geq \theta_{\mathcal{R}}, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

For monotonic functions, any threshold satisfies the diagonal quadrant property. However, a reasonable threshold choice increases the accuracy of the conversion algorithm—if one chooses the threshold $\sigma_{\mathbb{R}}(-10000) \approx 0$ then the BG will almost certainly always yield true.

Threshold on the Domain ($\theta_{\mathcal{D}}$). Since this work is limited to activation functions satisfying the Diagonal Quadrant Property, the threshold can be uniquely identified by only considering the domain. Once the domain's threshold is found, the activation function and range threshold can be effectively discarded. The more computationally efficient (and equivalent) definition of $\sigma_{\mathbb{B}}$ is then:

$$\sigma_{\mathbb{B}}(\theta; \theta_{\mathcal{D}}) = \begin{cases} 1 & \text{if } \theta \geq \theta_{\mathcal{D}}, \\ 0 & \text{otherwise.}^2 \end{cases} \quad (4)$$

Henceforth, $\theta_{\mathcal{D}} = 0$ and when $\sigma_{\mathbb{B}}$ is used without all parameters, the implicit parameters are $(\theta; 0)$.

5 DATA STRUCTURE

5.1 Neural Network Data Structure

The neural network, NN, is defined as an array of layers.

$$NN \leftarrow [l_1, l_2, \dots, l_m] \quad (5)$$

²Since $\sigma_{\mathbb{B}}(\theta_{\mathcal{D}})$ is undefined, letting $\sigma_{\mathbb{B}}(\theta_{\mathcal{D}}) = 0$ or $\sigma_{\mathbb{B}}(\theta_{\mathcal{D}}) = 1$ is arbitrary or dependent on special cases.

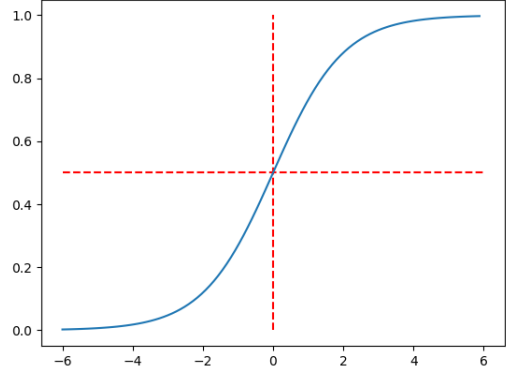


Figure 3: $\sigma_{\mathcal{R}}(\theta)$ with a threshold of $\sigma_{\mathcal{R}}(0) = 0.5$ satisfies the diagonal quadrants property.

Each layer, l_i , is an array of nodes.

$$l_i \leftarrow [\text{node}_{i,0}, \text{node}_{i,1}, \dots, \text{node}_{i,n}] \quad (6)$$

Each $\text{node}_{i,j}$ is an array of in-degree weights where $w_{i,j,b}$ is the bias weight and $w_{i,j,k}$ is the weight from $\text{node}_{i-1,k}$ to $\text{node}_{i,j}$.

$$\text{node}_{i,j} \leftarrow [w_{i,j,b}, w_{i,j,0}, w_{i,j,1}, \dots, w_{i,j,p}] \quad (7)$$

In the context of the current $\text{node}_{i,j}$, the bias weight is expressed as: $b = w_{i,j,b}$.

Output Definitions. out_i is the array of binarized outputs for layer l_i and $o_{i,j}$ is the binarized output of $\text{node}_{i,j}$.

$$\text{out}_i \leftarrow [o_{i,0}, o_{i,1}, \dots, o_{i,n}] \quad (8)$$

Since the neural network uses the $\sigma_{\mathcal{R}}$ activation function (Figure 3), we use the $\sigma_{\mathbb{B}}$ binary-step function as defined in Section 4 with the threshold $\theta_{\mathcal{D}} = 0$. Note that the θ in activation functions $f(\theta)$ is actually a dependent variable:

$$\theta = \theta(\vec{x}, \vec{w}, b) = \vec{x} \cdot \vec{w} + b. \quad (9)$$

The inputs $\vec{x} = \text{out}_{i-1}$, the weights $\vec{w} = \text{node}_{i,j} \setminus w_{i,j,b}$, and the bias $b = w_{i,j,b}$.

$$o_{i,j} = \sigma_{\mathbb{B}}(\theta(\text{out}_{i-1}, \text{node}_{i,j} \setminus b, b)) = \begin{cases} 1 & \text{if } \theta \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

The first layer, l_0 , has no in-degree weights and out_0 is the binary inputs for the neural network.

6 HEURISTICS AND PROPAGATION

6.1 Neural Constantness Heuristic (NCH)

The Neural Constantness Heuristic (NCH) checks for a neural node's constantness with exactly linear computational complexity where n is the number of in-degree weights for a given node. To understand this heuristic, let us create some variables and functions. For node $_{i,j}$:

- Let b be the bias weight $w_{i,j,b}$.
- Let W^+ be the set of weights greater than zero in node $_{i,j} \setminus b$
- Let $GV_{i,j}$ be the largest possible combination of weights in node $_{i,j}$.

$$GV_{i,j} = b + \sum_{w \in W^+} w \quad (11)$$

- Let W^- be the set of weights less than zero in node $_{i,j} \setminus b$.
- Let $SV_{i,j}$ be the smallest possible combination of weights in node $_{i,j}$.

$$SV_{i,j} = b + \sum_{w \in W^-} w \quad (12)$$

As seen above, b is always true and must be considered when computing both $GV_{i,j}$ and $SV_{i,j}$ regardless of its positivity. A heuristic to find the binarized output for a neural node is then:

$$o_{i,j} \leftarrow \begin{cases} 1 & \text{if } SV_{i,j} \geq 0, \\ 0 & \text{if } GV_{i,j} \leq 0 \\ \mathcal{B} & \text{otherwise.} \end{cases} \quad (13)$$

The logic is if the greatest possible value of θ is less than zero, then \mathcal{B} will produce a constant false. Alternatively, if the smallest possible value is greater than zero \mathcal{B} yields true. Otherwise, \mathcal{B} approximates the node.

6.2 Neural Real-valued Constant Heuristic (NRVCH)

Here, we extend NCH to approximate nodes as real-valued instead of binary. The Neural Real-Valued Constant Heuristic (NRVCH) has less than or equal to \mathcal{B} with NCH's sum-squared error. Furthermore, we find that NRVCH elicits many more constant nodes than NCH. Since NRVCH assumes an input space in

$\{0,1\}^n$, it must use \mathcal{N} . In addition to making the same assumptions as NCH in Section 6.1, NRVCH assumes the activation function is monotonically non-decreasing (as most activation functions are). We assume $f = \sigma$; however, NRVCH can be extended to other functions.

To determine if a node should be approximated as constant, NRVCH considers a maximum threshold on the range, $\alpha_{\mathcal{R}}$. Then the difference between range extremes of the weights in node $_{i,j}$ are tested to be less than or equal to $\alpha_{\mathcal{R}}$ where $\alpha_{\mathcal{R}} \leq 0.5$. If the condition is satisfied, the entire node is estimated as $f(avg_{\mathcal{D}})$ where $avg_{\mathcal{D}}$ is the average of all 2^n possible combinations of weights.

$$NRVCH: o_{i,j} \leftarrow \begin{cases} f(avg_{\mathcal{D}}) \\ \text{if } f(GV_{i,j}) - f(SV_{i,j}) \leq \alpha_{\mathcal{R}} \\ \mathcal{B} \text{ otherwise.} \end{cases} \quad (14)$$

We know that $f(LV_{i,j})$ and $f(GV_{i,j})$ yield the range extrema of f given node $_{i,j}$ since f is monotonic.

Average on the Domain ($avg_{\mathcal{D}}$). $avg_{\mathcal{D}}$ is chosen to sufficiently approximate the average over the range, $avg_{\mathcal{R}}$, in linear time and to minimize the sum-squared error (SSE).

For $\alpha_{\mathcal{R}} = 0.5$, the worst difference between $avg_{\mathcal{R}}$ and $f(avg_{\mathcal{D}})$ occurs with a weight vector $[0,0.5]$ given $f = \sigma$. Here, $\sigma(avg_{\mathcal{D}}) = 0.5621$ and $avg_{\mathcal{R}} = 0.5612$.

While $avg_{\mathcal{R}}$ gives less error than $f(avg_{\mathcal{D}})$, finding $avg_{\mathcal{R}}$ is in $O(w2^w)$ where w is the number of weights. In contrast, we obtain $avg_{\mathcal{D}}$ (linear to the number of weights) by realizing that the power set of $\{0,1\}$ always contains equal zeros and ones for each variable (see Table 1). Because of this, we can sum each weight, divide the summation by two, and then add the aggregated bias. Even when previous nodes are deemed constant and aggregated into the bias, node $_{i,j}$ is always in $\{0,1\}^n$. Substituting this formula for $avg_{\mathcal{D}}$,

$$o_{i,j} \leftarrow \begin{cases} f\left(\frac{w_{i,j,b} + \sum_{w \in \text{node}_{i,j}} w}{2}\right) \\ \text{if } f(GV_{i,j}) - f(SV_{i,j}) \leq \alpha_{\mathcal{R}} \\ \mathcal{B} \text{ otherwise.} \end{cases} \quad (15)$$

Theorem 1. $avg_{\mathcal{R}} \approx f(avg_{\mathcal{D}})$ minimizes the sum-squared error (SSE).

Table 1: The second Cartesian power of $\{0,1\}$ ($\{0,1\}^2$). Notice that 0 and 1 are evenly distributed among each variable x_i .

x_0	x_1
0	0
0	1
1	0
1	1

Proof. SSE is given by

$$E = \sum_{i=0}^n (x_i - \text{avg}_{\mathcal{R}})^2 \quad (16)$$

Where $[x_0, x_1, \dots, x_n]$ is the set of possible activation outputs.

$$\frac{\delta E}{\delta \text{avg}_{\mathcal{R}}} = -2 \sum_{i=0}^n (x_i - \text{avg}_{\mathcal{R}}) \quad (17)$$

$$\frac{\delta E}{\delta \text{avg}_{\mathcal{R}}} = 0 = \sum_{i=0}^n x_i - \sum_{i=0}^n \text{avg}_{\mathcal{R}} \quad (18)$$

$$\sum_{i=0}^n x_i = \sum_{i=0}^n \text{avg}_{\mathcal{R}} = n \times \text{avg}_{\mathcal{R}} \quad (19)$$

$$\text{avg}_{\mathcal{R}} = \frac{\sum_{i=0}^n x_i}{n} \quad (20)$$

So $\text{avg}_{\mathcal{R}}$ gives the minimum of E. \square

Additionally, given $\alpha_{\mathcal{R}} \leq 0.5$ and $f = \sigma$, NRVCH has less sum-squared error than NCH and creates constant node approximations at least as often as NCH.

6.3 Neural Constant Propagation

The constantness of previous nodes propagates throughout the network. While \mathcal{F} can inherently integrate this propagation, \mathcal{N} must take a preliminary forward traversal in linear time. We call this propagation of constants Neural Constant Propagation (NCP).

Propagation is done by aggregating constant nodes into the subsequent layer’s bias vector. As such, some nodes become constant that would not without NCP. For NCH, the constant nodes may only be represented as a 0 or 1, whereas NRVCH constants are in $(0,1)$.

Constant Aggregation Example using NRVCH.

Here, we give an example of a node that is only constant by NCP. Assume l_i has real-valued constant nodes $o_{i,1} = 0.7$ and $o_{i,2} = 0.5$. Then if node $_{i+1,0}$ is $[-1, 0, 3.2, 1]$ where -1 is the bias weight,

$$\text{node}_{i+1,0} \leftarrow [1.74, 0] \text{ since} \quad (21)$$

$$[1.74, 0] = [-1 + 0.7 \times 3.2 + 0.5 \times 1, 0] \quad (22)$$

Now, $SV_{i,j} \geq 0$ therefore $\sigma_{\mathcal{R}}(SV_{i,j}) \geq 0.5$. For $\sigma_{\mathcal{R}}$, $f(GV_{i,j})$ must be less than 1 so

$$f(GV_{i,j}) - f(SV_{i,j}) \leq 0.5 = \alpha_{\mathcal{R}}. \quad (23)$$

node $_{i+1,0}$ is now constant and will propagate to the next bias vector. Also, node $_{i+1,0}$ is now skipped in \mathcal{N} along with nodes node $_{i,1}$ and node $_{i,3}$.

7 NETWORK TRAVERSALS

In this section, we contrast two traversal algorithms: the forward traversal (\mathcal{F}) and the reverse traversal (\mathcal{N}).

\mathcal{N} discovers and skips negligible neural nodes entirely and ceases parsing at the input layer or upon finding a negligible layer. Both traversals benefit by the Neural Constant Propagation (NCP) that considers the previous layer’s constant nodes (Section 6.3). However, \mathcal{F} benefits more from NCP since \mathcal{F} finds more constant nodes due to entirely approximating l_{i-1} . To benefit from NCP, \mathcal{N} employs a preliminary forward traversal in linear time to find many of the constant nodes.

With NCP, \mathcal{N} improves average complexities for layers whose input space has a ratio of $x : T$ (number of constant nodes : total number of nodes) from $O(2^{0.5n}n)$ to $O(2^{0.5n(1-x)}n)$ where $0 \leq 1-x \leq 1$. Similarly, \mathcal{F} can also reduce the time complexity by omitting weights—however, the NN to BG algorithms mentioned in this paper cannot take full advantage of \mathcal{F} ’s weight pruning.

7.1 \mathcal{F} : Forward Traversal

\mathcal{F} approximates each node as a Boolean function one layer at a time from the input to the output layer.

In contrast to \mathcal{N} , \mathcal{F} has a finer understanding of the current layer’s input space (l_i) since the previous layer (l_{i-1}) has been approximated. We call this type of knowledge *shared logic*.

For example, approximating node $_{3,0}$ (from Figure 1) without any reduction has 2^n inputs (see Table 2). However, applying \mathcal{F} to Figure 1 up to node $_{3,0}$ finds that $o_{2,0} = o_{2,1}$. Therefore, the input space for l_3 is reduced by half to $\{[0,0], [1,1]\}$ since $[0,1]$ and $[1,0]$ can never occur. Applying \mathcal{B} to the reduced input space, we find that $o_{3,0} = 1$. See the reduced space in Table 3.

This technique can be extended to greater complexities. For example, if $\text{out}_2 = [o_{2,0}, o_{2,1}, o_{2,2}]$ and \mathcal{F} deduces that $o_{2,0} = o_{2,1} + o_{2,2}$, then the input space for l_3 can be reduced from 2^3 to 2^2 by removing the

Table 2: Exhaustive traversal space for node_{3,0}.

$o_{2,0}$	$o_{2,1}$	θ	$o_{3,0}$
0	0	1	1
0	1	2	1
1	0	1.1	1
1	1	2.1	1

Table 3: The reduced input space for node_{3,0} using \mathcal{F} 's shared logic.

$o_{2,0}$	$o_{2,1}$	θ	$o_{3,0}$
0	0	1	1
1	1	2.1	1

four impossible instances: $[1, 0, 0]$, $[0, 1, 1]$, $[0, 1, 0]$, and $[0, 0, 1]$.

Note that the reduced input space does not always yield an equivalent expression as shown above. The input space is reduced by asserting logical statements over primitive inputs (e.g. $o_{2,0} = o_{2,1} + o_{2,2}$). This maintains or reduces the number of true outputs in the Lookup table (LUT) which may alter the Boolean logic. However, the logic is effectively the same considering that only the impossible relations were omitted.

Shared-logic Setbacks. The shared logic advantage of \mathcal{F} has three notable setbacks. First, finding the shared logic may exhibit diminishing returns. Second, while reducing the available input space can yield simpler Boolean approximations, post-hoc Boolean simplification can achieve the same result. Third, the logic table may be shortened, but the instances of \mathcal{B} mentioned in this paper do not inherently take advantage of input spaces not in $\{0, 1\}^n$. Future work can solve, mitigate, and/or balance these setbacks.

7.2 \mathcal{N} : Reverse Traversal

\mathcal{N} (see Algorithm 1) does NCP in a single forward traversal using the heuristics NCH or NRVCH in linear time. Afterward, \mathcal{N} finds more negligible nodes and approximates the NN in its reverse traversal. This is done by establishing which outputs, out_{i-1} are used in each $node_{i,j} \in l_i$. If no nodes in the previous layer are used, they are deemed negligible and remembered in the DoPrevInputsMatter data structure.

Shared Logic. Detecting shared logic to the extent of \mathcal{F} without approximating the entire node (thus defeating the purpose for \mathcal{N}) is left for future work.

Algorithm 1: \mathcal{N} : Reverse traversal. Here, \mathcal{B} integrates the heuristics (e.g. NRVCH).

Input:

- $NN = [l_1, l_2, \dots, l_m]$.
- $\theta_{\mathcal{D}}$: the threshold on the domain.

Output: NNBG: a BG approximation of the neural network.

$ct.out = [out_0, out_1, \dots, out_{i-1}, \dots, out_{m-1}] \leftarrow$
reduced l_i input spaces using NCP;

$ct.bias \leftarrow$ aggregated biases from NCP via summation;

/* Each node in the final layer
always matters. */

DoCurrentInputsMatter $\leftarrow [true_0, true_1, \dots,$
 $true_{length(l_m)}]$;

for $l_i \leftarrow l_m, l_{m-1}, \dots, l_1$ **do**

inputSpace $\leftarrow ct.out_{i-1}$;

forall $node_{i,j} \in l_i$ **do**

if DoCurrentInputsMatter[j] **then**

 DoPrevInputsMatter, $o_{i,j} \leftarrow$

$\mathcal{B}(node_{i,j}, \theta_{\mathcal{D}}, ct.bias_{i,j}, inputSpace)$;

LayerBG $_i \leftarrow o_{i,0}|o_{i,1}| \dots |o_{i,n}$;

if true is not in DoPrevInputsMatter **then**

break;

DoCurrentInputsMatter \leftarrow
DoPrevInputsMatter;

NNBG \leftarrow aggregate layers sequentially;

return NNBG

7.3 Traversal Comparison

Let's compare \mathcal{F} and \mathcal{N} with the neural network in Figure 1. Here, \mathcal{N} only parses the output node with three weights while \mathcal{F} parses all hidden nodes and the output node for a total of fifteen weights. \mathcal{N} creates a more concise BG than \mathcal{F} since \mathcal{F} recalls negligible node logic. Both traversals implement a Boolean simplification algorithm after parsing before translation to a BG.

8 CONCLUSION AND FUTURE WORK

8.1 Summary

We successfully introduce heuristics (Section 6) and two traversal techniques (Section 7) the Neural Constant Heuristic (NCH), the Neural Real-Valued Con-

stant Heuristic (NRVCH), the Neural Constant Propagation (NCP), the forward traversal (\mathcal{F}), and the reverse traversal (\mathcal{N}).

NCH is functionally equivalent to \mathcal{B} (a generic NN to BG algorithm). NRVCH produces results different from \mathcal{B} and produces at most as much sum-squared error as NCH. Furthermore, NRVCH translates at least as many nodes to constant values as NCH. Both heuristics allow some nodes to be calculated in linear time.

NCP uses constant nodes from previous layers to reduce the weight space in the current layer. The propagation technique is better implemented with NRVCH but can be done with NCH.

\mathcal{F} uses its perfect knowledge of the previous layer to reduce the current layer’s input space via shared logic and does not complement most \mathcal{B} algorithms. In contrast, \mathcal{N} suits many \mathcal{B} options and omits neural nodes or layers entirely.

All things considered, the union of NRVCH, NCP, and \mathcal{N} is often the best choice for computational complexity, conciseness, and accuracy.

8.2 Future Work

Immediately following this paper, research can prove that NRVCH is viable for a larger set of activation functions than described here. Moreover, the average complexity improvement of these heuristics and traversals should be investigated (given the “average” neural network (NN)). An approximate complexity for the general case is likely too broad, and several subsets of networks given separate hyperparameters should be considered and specifically addressed. Consequently, related research can investigate what neural networks and data sets are most susceptible to constant neural nodes. Other potential work includes finding ways to leverage the shared logic found in \mathcal{F} with \mathcal{N} .

In broader disciplines, one can incorporate traditional neural network pruning with the approaches presented here. Or one could use this work for transfer learning by extracting Boolean logic from two binary neural networks, combining the logic, then mapping the combined logic to a new network.

ACKNOWLEDGEMENTS

This paper is partially funded by the AFRL Research Grant FA8650-20-F-1956.

REFERENCES

- Andrews, R., Diederich, J., and Tickle, A. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 6:373–389.
- Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., and Müller, K.-R. (2010). How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831.
- Brayton, R. and Mishchenko, A. (2010). Abc: An academic industrial-strength verification tool. volume 6174, pages 24–40.
- Briscoe, J. (2021). *Comprehending Neural Networks via Translation to And-Inverter Graphs*.
- Brudermueller, T., Shung, D., Laine, L., Stanley, A., Laursen, S., Dalton, H., Ngu, J., Schultz, M., Stegmaier, J., and Krishnaswamy, S. (2020). Making logic learnable with neural networks.
- Chan, H. and Darwiche, A. (2012). Reasoning about bayesian network classifiers. *arXiv preprint arXiv:1212.2470*.
- Choi, A., Shi, W., Shih, A., and Darwiche, A. (2017). Compiling neural networks into tractable boolean circuits. *intelligence*.
- Danks, D. and London, A. J. (2017). Regulating autonomous systems: Beyond standards. *IEEE Intelligent Systems*, 32(1):88–91.
- Fiesler, E. (1992). Neural network formalization. Technical report, IDIAP.
- Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys*, 51.
- Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.
- Kingston, J. K. (2016). Artificial intelligence and legal liability. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 269–279. Springer.
- Kroll, J. A., Barocas, S., Felten, E. W., Reidenberg, J. R., Robinson, D. G., and Yu, H. (2016). Accountable algorithms. *U. Pa. L. Rev.*, 165:633.
- Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *CoRR*, abs/1710.05941.
- Shi, W., Shih, A., Darwiche, A., and Choi, A. (2020). On tractable representations of binary neural networks.