On the Relative Expressiveness of Bayesian and Neural Networks*

Arthur Choi[a], Ruocheng Wang[b], Adnan Darwiche[a]

[a] *Computer Science Department, University of California, Los Angeles, USA*
[b] *College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China*

Abstract

A neural network computes a function. A central property of neural networks is that they are "universal approximators:" for a given continuous function, there exists a neural network that can approximate it arbitrarily well, given enough neurons (and some additional assumptions). In contrast, a Bayesian network is a model, but each of its queries can be viewed as computing a function. In this paper, we identify some key distinctions between the functions computed by neural networks and those by marginal Bayesian network queries, showing that the former are more expressive than the latter. Moreover, we propose a simple augmentation to Bayesian networks (a testing operator), which enables their marginal queries to become "universal approximators."

Keywords: Bayesian networks, neural networks, arithmetic circuits, function approximation

1. Introduction

The field of artificial intelligence (AI) has seen two major milestones throughout its history. Shortly after the field was born in the 1950s, the focus turned to *symbolic, model-based* approaches, which were premised on the need to represent and reason with domain knowledge, and exemplified by the use of logic to represent such knowledge (McCarthy, 1959). In the 1980s, the focus turned to *probabilistic, model-based* approaches, as exemplified by Bayesian networks and probabilistic graphical models more generally (first major milestone) (Pearl, 1988). Starting in the 1990s, and as data became abundant, these probabilistic models provided the foundation for much of the research in machine learning, where models were learned either generatively or discriminatively from data. Recently, the field shifted its focus to *numeric, function-based* approaches, as exemplified by neural networks, which are trained discriminatively using labeled data (deep learning, second major milestone) (Goodfellow et al., 2016; Hinton et al., 2006; Bengio et al., 2006; Ranzato et al., 2006; Rosenblatt, 1958; McCulloch & Pitts, 1943). Perhaps the biggest surprise with the second milestone is the extent to which certain tasks, associated with perception or limited forms of cognition, can be approximated using functions (i.e., neural networks) that are learned purely from labeled data, without the need for modeling or reasoning (Darwiche, 2018).

While this evolution of the field has increased our abilities, the emerging techniques have been pursued by somewhat independent research communities. The price has been a lack of enough integration and fusion of the various methods, which hinders the realization of their collective benefits. *Logic* provides a rich framework for representing knowledge in the form of domain constraints and comes with profound reasoning mechanisms. *Probabilistic graphical models* excel at capturing uncertainty, causal knowledge, and independence information. Both of these frameworks provide a foundation for capturing domain knowledge of various types and for reasoning with such knowledge. *Neural networks* are effective function approximators,

*This article is an extended and revised version of Choi & Darwiche (2018), with more theoretical and experimental results.

Email addresses: aychoi@cs.ucla.edu (Arthur Choi), rchwang@outlook.com (Ruocheng Wang), darwiche@cs.ucla.edu (Adnan Darwiche)

allowing one to approximate specific and narrow tasks by simply fitting a complex function (i.e., deep neural network) to data in the form of input-output pairs—again, without the need for modeling or reasoning.

Each of these frameworks has its shortcomings though. Symbolic models are too coarse to capture certain phenomena and, in their pure form, miss out on exploiting data and the wealth of information it may contain. Probabilistic graphical models, in their generative form, address this limitation especially when they integrate symbolic knowledge. However, in their discriminative form, these models have been outperformed by neural networks as a realization of the function-based approach to AI. While most of the recent success in AI has been due to neural networks, we are now starting to realize their limits too: they are data hungry, may not generalize beyond the given data, can be quite brittle, and pose challenges for explanation and verification. Interestingly, it is these shortcomings that models, whether symbolic or probabilistic, can help alleviate.

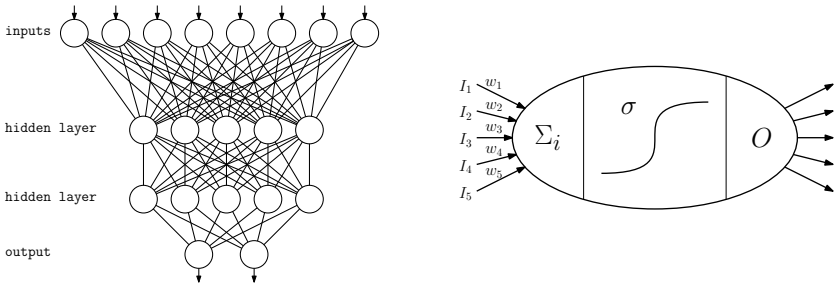
Hence, a key challenge and opportunity for AI today lies in *fusing* these approaches to realize their collective benefits.[1] We take some initial steps towards this goal in this paper by showing how models can empower the function-based approach to AI. We focus on probabilistic graphical models in the form of Bayesian networks, but our interest is ultimately in models that combine probabilistic and symbolic knowledge; see, e.g., (Shen et al., 2018; Kisa et al., 2014; Poole, 2003; Halpern, 1990; Nilsson, 1986). We start with a set of observations that are known in the literature but that together lead to a dilemma. Our main contribution is a resolution to this dilemma, which we argue has major implications on the quest of fusing model-based and function-based approaches to AI.

Our observations are as follows. First, a query posed to a Bayesian network model can be viewed as inducing (and evaluating) a function, which can be represented using an Arithmetic Circuit (AC) (Darwiche, 2003; Choi & Darwiche, 2017). Next, Bayesian network queries (and, hence, ACs) can be trained discriminatively using labeled data and gradient descent methods, leading to a realization of the function-based approach currently practiced using neural networks—except that a neural network represents only one function, while a Bayesian network represents many functions (one for each query). Finally, the functions induced by Bayesian network queries can integrate background knowledge of various forms, suggesting a more principled, if not more sophisticated, function-based approach compared to neural networks. Now the dilemma: recent developments in AI indicate that neural networks outperform Bayesian networks, and probabilistic graphical models more generally, as a realization of the function-based approach, despite the ability of the latter to integrate background knowledge into the learned functions.

Our first contribution towards resolving this dilemma is highlighting the following observation: Bayesian and neural networks induce classes of functions that differ in expressiveness. It is known that neural networks are *universal approximators,* which means they can approximate any function to an arbitrary small error. However, the functions induced by marginal Bayesian network queries are less expressive as they correspond to multi-linear functions or quotients of such functions (this is also known but was never discussed in this context). Our second contribution: We address this expressiveness gap by proposing a simple extension to Bayesian networks and showing that this extension can also induce functions that are universal approximators. When the newly induced functions are represented by circuits and trained using labeled data, we obtain a function-based approach that is as expressive as neural networks, but that can also integrate the knowledge embodied in a Bayesian network. In other words, we can now synthesize function structures from models and then learn their parameters from labeled data. The resulting function-based approach can therefore benefit from what models have to offer: less dependence on data, improved generality and robustness, better prospects for explainability and the potential for more direct formal guarantees on behavior.

This paper is structured as follows. We review in Section 2 the class of functions induced by neural and Bayesian networks, while identifying the corresponding gap in expressiveness. We then propose a new class of Bayesian networks in Section 3, *Testing Bayesian Networks (TBNs),* whose queries can be computed using *Testing Arithmetic Circuits (TACs)* that we discuss in Section 4. We show in Section 5 that these functions

[1] We distinguish between integration and fusion. *Integration* may refer to an intelligent agent architecture in which the components are based on different approaches, but work together to complement each other. *Fusion* may refer to a model-based approach that is empowered by functions, or a function-based approach that is empowered by models. Our focus in this paper is on fusion, particularly the empowerment of function-based approaches with domain knowledge (i.e., models).



(a) A neural network structure (b) A mathematical model of a neuron

Figure 1: A neural network and a neuron.

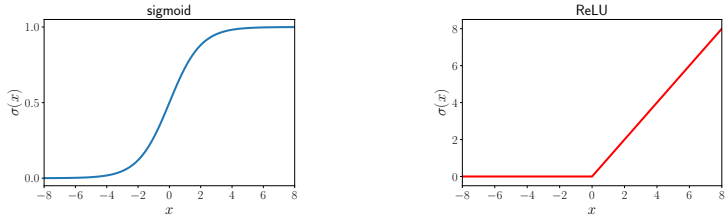


Figure 2: Two activation functions. A sigmoid $\sigma(x) = \frac{1}{1+\exp\{-x\}}$ acts as a soft threshold which tends to 0 as x goes to $-\infty$ and tends to 1 as x goes to ∞. A ReLU $\sigma(x) = \max(0, x)$ is equal to 0 if $x < 0$ and is equal to x otherwise.

are universal approximators and reveal their specific functional form in Section 6. We then show some experimental results in Section 7 and conclude in Section 8. The appendices include the method we used to train TACs from labeled data, the full proof of the universal approximation theorem, and an algorithm for compiling TBN queries into TACs.

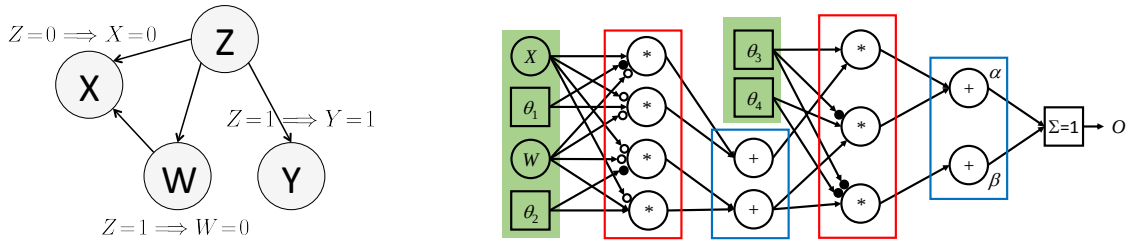
2. Technical Background

We next review the class of functions represented by neural networks. We also highlight previous results, allowing us to view Bayesian network queries as inducing and evaluating functions. The goal is to pinpoint an expressiveness gap between the two classes of functions, which we address in Section 3.

2.1. Neural Networks as Functions

A (feedforward) neural network is a directed acyclic graph (DAG); see Figure 1(a). The roots of the DAG are the neural network inputs, call them $X_1, \ldots, X_n$. The leaves of the DAG are the neural network outputs, call them $Y_1, \ldots, Y_m$. Each node in the DAG is called a *neuron* and contains an *activation function* σ; see Figure 1(b). Each edge I in the DAG has a *weight* w attached to it. The weights of a neural network are its *parameters,* which are learned from data. Consider a neuron with activation function σ, inputs I_i and corresponding weights w_i. The output of this neuron is simply $\sigma(\sum_i w_i \cdot I_i)$. Thus, one can compute the output Y_j of a neural network by simply evaluating neurons, parents before children, which can be done in time linear in the neural network size.

To simplify the discussion, we will assume that a neural network has a single output Y. Hence, a neural network represents a function $Y = f(X_1, \ldots, X_n)$. A key question here relates to the class of functions that can be represented, or approximated well, by neural networks that use a certain class of activation functions. One example is the *sigmoid* activation function; see Figure 2. A neural network with only sigmoid activation



(a) A Bayesian network with logical constraints (0/1 parameters)

(b) A function that computes the query $O = P(y|x, w)$. Here, $\alpha = P(y, x, w)$ and $\beta = P(\bar{y}, x, w)$

Figure 3: The function uses adders (+), multipliers (∗), inverters (○), $1-\theta$ units (●), and normalizing units ($\Sigma = 1$). Excluding the Σ unit (division), and emulating ○ and ● units using adders, we obtain an Arithmetic Circuit (AC) (Darwiche, 2003; Choi & Darwiche, 2017). The function implicitly integrates the Bayesian network's 0/1 parameters. Moreover, $\theta_1, \ldots, \theta_4$ are Bayesian network parameters that have specific interpretations.

functions can approximate any continuous function to within an arbitrary error.[2] Such neural networks are called *universal approximators* of continuous functions (Hornik et al., 1989; Cybenko, 1989; Leshno et al., 1993). Most of the recent neural networks are based on the *ReLU* activation function, which is simpler than the sigmoid; see Figure 2. Neural networks with ReLUs are also universal approximators of continuous functions (Leshno et al., 1993).

2.2. Bayesian Network Queries as Functions

A Bayesian network is a directed acyclic graph (DAG), where each node is associated with a conditional probability table (CPT). The CPT for node X with parents $\mathbf{U}$ contains a parameter $\theta_{x|\mathbf{u}} \in [0, 1]$ for each state x of node X and state $\mathbf{u}$ of parents $\mathbf{U}$, such that $\sum_x \theta_{x|\mathbf{u}} = 1$. A Bayesian network specifies a unique probability distribution over its nodes (Pearl, 1988).

Consider now a Bayesian network, some evidence $\mathbf{e}$ on variables $\mathbf{E}$ (e.g., symptoms), and let Y be a query variable (e.g., a disease). The probability $P(y, \mathbf{e})$ can be viewed as the output of a function $f(\mathbf{E})$ that maps evidence $\mathbf{e}$ into a number in $[0, 1]$. The function inputs are discrete values of variables $\mathbf{E}$, but can be continuous values in $[0, 1]$ if one uses soft evidence (Chan & Darwiche, 2005) (universal approximation results for neural networks assume that inputs/outputs are in $[0, 1]$).

The function $f(\mathbf{E})$ can be represented by an *Arithmetic Circuit (AC)* containing only multipliers and adders (Darwiche, 2003; Choi & Darwiche, 2017); see Figure 3. Classical inference algorithms for Bayesian networks implicitly construct and evaluate this circuit on the fly (Darwiche, 2009, Chapter 12), while other approaches explicitly construct this circuit through a compilation process (Darwiche, 2003, 2009). Consider now the function corresponding to a Bayesian network query $P(\alpha)$. It is known that this function is *multi-linear* (more on this later), regardless of the query α and the underlying Bayesian network (Darwiche, 2003, 2009).[3] Hence, the functions induced by Bayesian network queries (and ACs) are less expressive than the ones represented by neural networks. This can explain why a Bayesian network that is trained discriminatively may not outperform a neural network when the labeled data is generated by a function that is not multi-linear.[4]

Here is our major insight for addressing this expressiveness gap, which is based on a simple but consequential observation. It is known that if each activation function of a neural network is a polynomial, then

[2] A *shallow* neural network with a single hidden layer is sufficient for universal approximation, but may require exponentially many neurons. A *deep* neural network may be more succinct for this purpose though.

[3] The conditional probability $P(\alpha|\mathbf{e})$ is the quotient of two multi-linear functions.

[4] Refined representations of CPTs do not change the expressiveness of functions induced by Bayesian networks. For example, in a *sigmoid belief network*, the CPTs of nodes are represented as log linear models (Neal, 1992), i.e., like a neuron with a sigmoid activation. These "sigmoid" CPTs are compact representations of tabular CPTs and facilitate the development of learning algorithms (Neal, 1992; Saul et al., 1996), but do not increase expressiveness.

the neural network can only represent polynomials (Leshno et al., 1993). Moreover, neural networks with linear activation functions can only represent linear functions. Consider now the ReLU activation function $\sigma(x) = \max(0, x)$, which leads to a universal approximator. This function equals 0 if $x < 0$ and equals x otherwise; see Figure 2. Hence, it is two linear functions augmented with a simple *test,* $x < 0$, for *selecting* one of them. What this tells us is that we can turn ACs into universal approximators by only augmenting them with *testing units.* We show this later, leading to *Testing Arithmetic Circuits (TACs).* In fact, the notion of testing can be integrated directly into Bayesian networks, leading to *Testing Bayesian Networks (TBNs).* A TAC computes a TBN query, just like an AC computes a Bayesian network query.

3. Testing Bayesian Networks

The concept of a Testing Bayesian Network (TBN) is relatively simple. In a nutshell: it is a Bayesian network whose CPTs are selected dynamically based on the given evidence.

Consider a Bayesian network that contains a binary node X having a single binary parent U. The CPT for node X contains *one* distribution on X for each state u of its parent:

U	X	
u	x	$\theta_{x\|u}$
u	$\bar{x}$	$\theta_{\bar{x}\|u}$
$\bar{u}$	x	$\theta_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$	$\theta_{\bar{x}\|\bar{u}}$

In a TBN, node X may be *testing.* In this case, we need *two* distributions on X for each state u of its parent. Moreover, we need a *threshold* for each state u, which is used to select one of these distributions:

U	X			
u	x	T_u	$\theta^+_{x\|u}$	$\theta^-_{x\|u}$
u	$\bar{x}$		$\theta^+_{\bar{x}\|u}$	$\theta^-_{\bar{x}\|u}$
$\bar{u}$	x	$T_{\bar{u}}$	$\theta^+_{x\|\bar{u}}$	$\theta^-_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$		$\theta^+_{\bar{x}\|\bar{u}}$	$\theta^-_{\bar{x}\|\bar{u}}$

The selection of distributions utilizes the posterior on parent U given evidence on X's *ancestors.* For parent state u, the selected distribution on X is $(\theta^+_{x|u}, \theta^+_{\bar{x}|u})$ if the posterior on u is $\geq T_u$; otherwise, it is $(\theta^-_{x|u}, \theta^-_{\bar{x}|u})$. For parent state $\bar{u}$, the distribution is $(\theta^+_{x|\bar{u}}, \theta^+_{\bar{x}|\bar{u}})$ if the posterior on $\bar{u}$ is $\geq T_{\bar{u}}$; otherwise, it is $(\theta^-_{x|\bar{u}}, \theta^-_{\bar{x}|\bar{u}})$. Thus, the CPT for node X is determined *dynamically* based on the two thresholds and the posterior over parent U, leading to one of the following four CPTs:[5]

U	X	
u	x	$\theta^+_{x\|u}$
u	$\bar{x}$	$\theta^+_{\bar{x}\|u}$
$\bar{u}$	x	$\theta^+_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$	$\theta^+_{\bar{x}\|\bar{u}}$

U	X	
u	x	$\theta^+_{x\|u}$
u	$\bar{x}$	$\theta^+_{\bar{x}\|u}$
$\bar{u}$	x	$\theta^-_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$	$\theta^-_{\bar{x}\|\bar{u}}$

U	X	
u	x	$\theta^-_{x\|u}$
u	$\bar{x}$	$\theta^-_{\bar{x}\|u}$
$\bar{u}$	x	$\theta^+_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$	$\theta^+_{\bar{x}\|\bar{u}}$

U	X	
u	x	$\theta^-_{x\|u}$
u	$\bar{x}$	$\theta^-_{\bar{x}\|u}$
$\bar{u}$	x	$\theta^-_{x\|\bar{u}}$
$\bar{u}$	$\bar{x}$	$\theta^-_{\bar{x}\|\bar{u}}$

In general, if the parents of testing node X have n states, the selection process may yield 2^n distinct CPTs. We will now give two illustrative examples of TBNs before we define their syntax and semantics formally.

Consider Figure 4(a), where all nodes are binary. Node Y is testing and has the following *testing CPT:*

H	Y			
h	y	$T_h = t$	$\theta^+_{y\|h} = 1$	$\theta^-_{y\|h} = 0$
h	$\bar{y}$		$\theta^+_{\bar{y}\|h} = 0$	$\theta^-_{\bar{y}\|h} = 1$
$\bar{h}$	y	$T_{\bar{h}} = 1 - t$	$\theta^+_{y\|\bar{h}} = 0$	$\theta^-_{y\|\bar{h}} = 1$
$\bar{h}$	$\bar{y}$		$\theta^+_{\bar{y}\|\bar{h}} = 1$	$\theta^-_{\bar{y}\|\bar{h}} = 0$

[5] In general, testing can take other forms such as $> T$, $\leq T$ or $< T$.

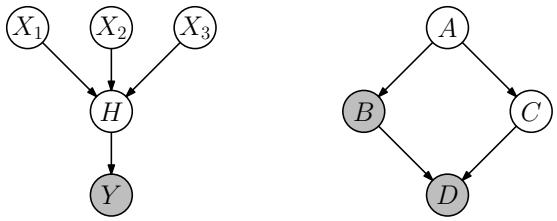


Figure 4: Two testing Bayesian Networks (TBNs). All nodes are binary. Testing nodes are shaded.

Suppose now we have evidence $\mathbf{e} = x_1, x_2, x_3$, which pertains to the ancestors of testing node Y. In this example, the selected CPT for node Y will be based on the tests $P(h \mid \mathbf{e}) \geq T_h$ and $P(\bar{h} \mid \mathbf{e}) > T_{\bar{h}}$. For parent state h, the selected distribution on $(y, \bar{y})$ is $(1, 0)$ if $P(h \mid \mathbf{e}) \geq T_h = t$; otherwise, it is $(0, 1)$. For parent state $\bar{h}$, the selected distribution on $(y, \bar{y})$ is $(0, 1)$ if $P(\bar{h} \mid \mathbf{e}) > T_{\bar{h}} = 1 - t$; otherwise, it is $(1, 0)$.

Nodes X_1, X_2, X_3 and H can be viewed as the basis of a noisy-or classifier as in Vomlel (2006). That is, we classify an instance x_1, x_2, x_3 positively iff $P(h|x_1, x_2, x_3) \geq t$, where t is the classification threshold. The TBN in Figure 4(a) can then be viewed as implementing this classifier since $P(y|x_1, x_2, x_3) = 1$ if instance x_1, x_2, x_3 is positive, and $P(y|x_1, x_2, x_3) = 0$ if the instance is negative.

More generally, we may have multiple testing nodes in a TBN. Figure 4(b) depicts a TBN with two testing nodes, B and D, where all variables are binary. In a classical Bayesian network, we need 18 parameters to fully specify the network: 2 for A, 4 for each of B, C and 8 for D. For the TBN, we need 30 parameters: 4 additional parameters for B and 8 additional parameters for D. We also need 2 thresholds for B and 4 thresholds for D.

From now on, we will use BN to denote a classical Bayesian network and TBN for a testing one.

3.1. TBN Syntax

A TBN is a directed acyclic graph (DAG) with two types of nodes: *regular* and *testing,* each having a conditional probability table (CPT). Root nodes are always regular. Consider a node X with parents $\mathbf{U}$.

- If X is a regular node, its CPT is said to be *regular* and has a parameter $\theta_{x|\mathbf{u}} \in [0, 1]$ for each state x of node X and state $\mathbf{u}$ of its parents $\mathbf{U}$, such that $\sum_x \theta_{x|\mathbf{u}} = 1$ (these are the CPTs used in BNs).
- If X is a testing node, its CPT is said to be *testing* and has a threshold $T_{X|\mathbf{u}} \in [0, 1]$ for each state $\mathbf{u}$ of parents $\mathbf{U}$. It also has two parameters $\theta^+_{x|\mathbf{u}} \in [0, 1]$ and $\theta^-_{x|\mathbf{u}} \in [0, 1]$ for each state x of node X and state $\mathbf{u}$ of its parents $\mathbf{U}$, such that $\sum_x \theta^+_{x|\mathbf{u}} = 1$ and $\sum_x \theta^-_{x|\mathbf{u}} = 1$.

The parameters of a regular CPT are said to be *static* and the ones for a testing CPT are said to be *dynamic.*

Consider a node that has m states and its parents have n states. If the node is regular, its CPT will have $m \cdot n$ static parameters. If it is a testing node, its CPT will have n thresholds and $2 \cdot m \cdot n$ dynamic parameters. As we shall discuss later, the thresholds and parameters of a TBN can be learned discriminatively from labeled data (as in deep learning).

3.2. TBN Semantics

A testing CPT corresponds to a set of regular CPTs, one of which is selected based on the given evidence. Once a regular CPT is selected from each testing CPT, the TBN transforms into a BN. In other words, a TBN over DAG G represents a set of BNs over DAG G, one of which is selected based on the given evidence. It is this selection process that determines the semantics of TBNs. We define this process next based on soft evidence, which includes hard evidence as a special case.[6]

[6]Choi & Darwiche (2018) used soft evidence on root nodes only, which is sufficient for the universal approximation theorem.

Soft evidence on a variable X with states $x_1, \ldots, x_k$ is specified using *likelihood ratios* $\lambda_1, \ldots, \lambda_k$ (Pearl, 1988). Without loss of generality, we require $\lambda_1 + \ldots + \lambda_k = 1$ so $\lambda_i = 1$ corresponds to hard evidence $X = x_i$. Moreover, when node X is binary, soft evidence reduces to a single number $\lambda_x \in [0, 1]$ since $\lambda_{\bar{x}} = 1 - \lambda_x$.

We will emulate soft evidence by hard evidence on an auxiliary, binary child S with the following CPT:

X	S	$\Theta_{S\|x}$
x_1	s	λ_1
x_1	$\bar{s}$	$1 - \lambda_1$
$\vdots$	$\vdots$	$\vdots$
x_k	s	λ_k
x_k	$\bar{s}$	$1 - \lambda_k$

We can now take $P(.|s)$ as the result of asserting soft evidence on node X since

$$\frac{P(x_1|s)}{P(x_1)} \; : \; \ldots \; : \; \frac{P(x_k|s)}{P(x_k)} = \lambda_1 \; : \; \ldots \; : \; \lambda_k.$$

Let $\mathbf{s}$ denote all available soft evidence. We will use $\mathcal{P}^\star(.)$ to denote the conditional distribution $P(.|\mathbf{s})$ and $P^\star(.)$ to denote the joint distribution $P(., \mathbf{s})$.

We now show how a TBN can be converted into a BN, thereby defining the semantics of TBNs. We start with an empty BN and traverse the TBN nodes, parents before children. Suppose we are visiting TBN node X with parents $\mathbf{U}$. If node X is regular, we add it to the BN as a child of nodes $\mathbf{U}$, while copying its regular CPT to the BN. If node X is testing, we first use the partially constructed BN to compute the posterior $\mathcal{P}^\star(\mathbf{U})$ using soft evidence on the ancestors of node X. Using this posterior, we then select a regular CPT for node X from its testing CPT, e.g., as follows:[7]

$$\theta_{x|\mathbf{u}} = \begin{cases} \theta^+_{x|\mathbf{u}} & \text{if } \mathcal{P}^\star(\mathbf{u}) \geq T_{X|\mathbf{u}} \\ \theta^-_{x|\mathbf{u}} & \text{otherwise.} \end{cases}$$

We finally add node X to the BN as a child of nodes $\mathbf{U}$ and copy its selected, regular CPT to the BN.

After visiting all nodes in the TBN, the constructed BN will have the same structure as the TBN. We can now use this BN to answer queries using all available evidence, as is normally done.

In this paper, we assume that testing nodes select their CPTs based on evidence on their *ancestors,* which is sufficient for our main result on universal approximation.[8] This assumption was recently relaxed though to allow the integration of more evidence into the selection process (Shen et al., 2019). It was also shown that conditional independence in TBNs can be inferred graphically using d-separation, just like in BNs (Shen et al., 2019). This is one additional example of how TBNs retain some of the model-based advantages of BNs despite acquiring the expressiveness of function-based approaches such as neural networks, which we will discuss later in Section 5.

4. Testing Arithmetic Circuits

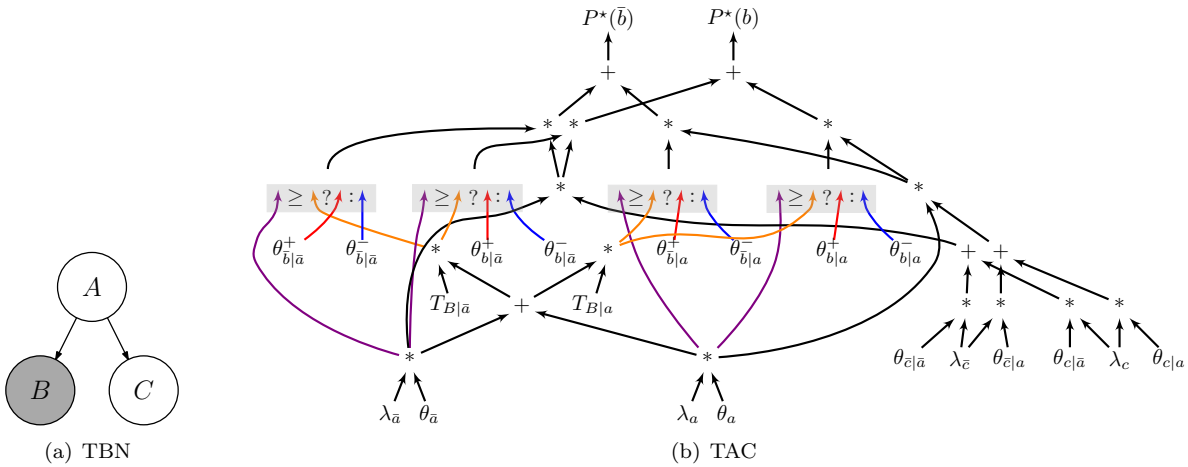
A Testing Arithmetic Circuit (TAC) is an Arithmetic Circuit (AC) that includes *testing units.* A testing unit has two inputs, x and T, and two parameters, θ^+ and θ^-. Its output is computed as follows:[9]

$$f(x, T) = \begin{cases} \theta^+ & \text{if } x \geq T \\ \theta^- & \text{otherwise.} \end{cases}$$

[7] The selection can be based on other tests such as $\mathcal{P}^\star(\mathbf{u}) > T_{X|\mathbf{u}}$, $\mathcal{P}^\star(\mathbf{u}) \leq T_{X|\mathbf{u}}$ or $\mathcal{P}^\star(\mathbf{u}) < T_{X|\mathbf{u}}$.

[8] The dependence of CPT selection on only ancestral evidence can be limiting practically since not all TBN queries can fully benefit from the power of CPT selection. In the extreme case of evidence laying below testing nodes in a TBN, CPT selection will not be impacted by the available evidence. CPT selection can be made to depend on more evidence, beyond ancestral, as long as it does not lead to selection ambiguities (i.e., the selection of one CPT impacting the selection of another). Resolving such potential ambiguities, however, requires a more thorough treatment.

[9] A testing unit may employ other tests such as $x > T$, $x \leq T$ or $x < T$.



(a) TBN (b) TAC

Figure 5: Nodes A, B and C are binary and node B is testing. Nodes $\boxed{x \geq T \,?\, \theta^+ : \theta^-}$ represent testing units.

Just like an AC computes a BN query (see Figure 3), a TAC computes a TBN query. Appendix C provides an algorithm for compiling a TAC that computes a given TBN query.

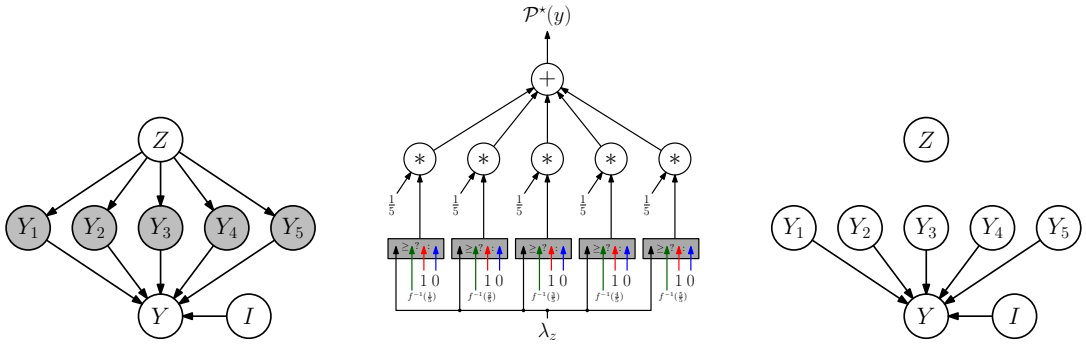
This algorithm was used to compile the TAC in Figure 5(b), which has four testing units and computes a query on the TBN in Figure 5(a). The TAC inputs $(\lambda_a, \lambda_{\bar{a}})$ and $(\lambda_c, \lambda_{\bar{c}})$ represent soft evidence on nodes A and C, respectively. Its outputs $P^\star(b)$ and $P^\star(\bar{b})$ represent the marginal distribution on node B. All other TAC inputs correspond to TBN parameters and thresholds: 2 static parameters for node A, 4 static parameters for node C, in addition to 8 dynamic parameters and 2 thresholds for node B.

As discussed earlier, TBNs are motivated by the recent success of neural networks in learning functions from labeled data. When using a neural network in this context, the function structure is usually handcrafted while its parameters are learned using gradient descent methods. While neural networks have been very successful in this context, they have also been subject to scrutiny due to their opaqueness and inability to integrate domain knowledge in a principled manner. Opaqueness makes neural networks hard to explain and verify. The inability to integrate domain knowledge diminishes their robustness and implies that we may need a massive amount of data to train them successfully.

The structure of a function computed by a TBN query is *compiled* from the TBN. This structure takes the form of a TAC, which parameters and thresholds can be learned from labeled data using gradient descent methods. The advantage of compiling a function structure from a model, in contrast to handcrafting it, is that we can integrate some forms of background knowledge into the function. For example, all the independence assumptions encoded by the TBN will be respected by the compiled TAC, regardless of how we train it. Moreover, some TBN parameters may already be known and these will be carried into the compiled TAC, without the need to train them from data, therefore reducing the dependence on data. This particularly includes 0/1 parameters, which correspond to logical domain constraints. Finally, since a TAC is compiled from a TBN, one stands a better chance at explaining and verifying the TAC behavior. One must note though that the expressiveness of a TAC is mandated by the underlying TBN query so it cannot be arbitrarily controlled, as in handcrafted neural networks. We discuss this issue in the next two sections.

5. Expressiveness of TBN Queries and TACs

A central concept relating to expressiveness is that of universal approximation. Consider the class of continuous functions $f(x_1, \ldots, x_n)$ from $[0,1]^n$ to $[0,1]$. A representation is said to be a universal approximator



(a) TBN (b) TAC for query $\mathcal{P}^\star(y)$ (c) BN after CPT selection

Figure 6: Approximating a continuous, monotonic function $f(x)$ using a TBN query for $N = 5$.

if it can approximate any function in this class to an arbitrary small error.[10] As mentioned earlier, neural networks with appropriate activation functions are universal approximators.

TBN queries and, hence, TACs are also universal approximators. The following theorem shows this for continuous and monotonic functions $f(x)$. The general (and slightly more involved) case of multivariate, non-monotonic functions is delegated to Appendix B.

Theorem 1. *Given a continuous, monotonic function $f(x)$ from $[0,1]$ to $[0,1]$ and error ε, there exists a TBN that contains $O(\lceil \frac{1}{\varepsilon} \rceil)$ nodes and that satisfies the following. The TBN contains two binary nodes, Z and Y, such that if x is the soft evidence on node Z, then $|\mathcal{P}^\star(y) - f(x) \leq \varepsilon|$.*

Proof Our constructive proof is based on Jones (1990) and uses the TBN in Figure 6(a). The TBN has regular nodes Z, Y and I, which are discrete. Nodes Z and Y are binary with states $z, \bar{z}$ and $y, \bar{y}$. Node I has values $1, \ldots, N$ and controls the quality of approximation (better approximation for larger N). The TBN has testing nodes $\mathbf{Y} = Y_1, \ldots, Y_N$, which are binary with states $y_i, \bar{y}_i$. The TBN query is to compute the probability of $Y = y$ given soft evidence on node Z. The TAC in Figure 6(b) computes this query.

Nodes Z and I have uniform priors. Node Y has the following CPT:

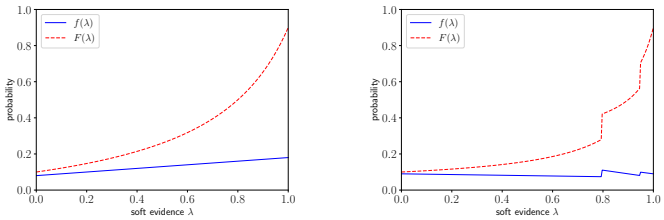
$$\theta_{y|\mathbf{y},i} = \begin{cases} 1 & \text{if } \mathbf{y} \text{ sets variable } Y_i \text{ to value } y \\ 0 & \text{otherwise} \end{cases}$$

That is, node Y is equivalent to node Y_i when $I = i$. The testing CPT for node Y_i is as follows:[11]

Z	Y_i							
z	y_i	$T_z =$	$f^{-1}(\frac{i}{N})$	$\theta^+_{y_i\|z} =$	1	$\theta^-_{y_i\|z} =$	0
z	$\bar{y}_i$			$\theta^+_{\bar{y}_i\|z} =$	0	$\theta^-_{\bar{y}_i\|z} =$	1
$\bar{z}$	y_i	$T_{\bar{z}} =$	$1 - f^{-1}(\frac{i}{N})$	$\theta^+_{y_i\|\bar{z}} =$	0	$\theta^-_{y_i\|\bar{z}} =$	1
$\bar{z}$	$\bar{y}_i$			$\theta^+_{\bar{y}_i\|\bar{z}} =$	1	$\theta^-_{\bar{y}_i\|\bar{z}} =$	0

Given soft evidence $\lambda_z = x$ on node Z, the posterior marginal $\mathcal{P}^\star(z)$ is then x. We can now select a regular CPT for each testing node Y_i, which must be one of[12]

[10] Typically, the function is assumed to be defined on a compact set (i.e., closed and bounded) and hence uniformly continuous.
[11] We assume, without loss of generality, that $f(0) = 0$ and $f(1) = 1$.
[12] The used tests are $\mathcal{P}^\star(z) \geq T_z$ and $\mathcal{P}^\star(\bar{z}) > T_{\bar{z}}$. We have $\mathcal{P}^\star(z) \geq T_z$ iff $1 - \mathcal{P}^\star(z) \leq 1 - T_z$ iff $\mathcal{P}^\star(\bar{z}) \leq T_{\bar{z}}$.



(a) BN function (b) TBN function

Figure 7: Functions induced by BN and TBN queries on the structure $E \to T \to Q$. Node T is testing in the TBN.

Z	Y_i	
z	y_i	$\theta_{y_i\|z} = 1$
z	$\bar{y}_i$	$\theta_{\bar{y}_i\|z} = 0$
$\bar{z}$	y_i	$\theta_{y_i\|\bar{z}} = 1$
$\bar{z}$	$\bar{y}_i$	$\theta_{\bar{y}_i\|\bar{z}} = 0$

Z	Y_i	
z	y_i	$\theta_{y_i\|z} = 0$
z	$\bar{y}_i$	$\theta_{\bar{y}_i\|z} = 1$
$\bar{z}$	y_i	$\theta_{y_i\|\bar{z}} = 0$
$\bar{z}$	$\bar{y}_i$	$\theta_{\bar{y}_i\|\bar{z}} = 1$

In either case, Y_i will no longer depend on Z, leading to the BN structure in Figure 6(c). Using θ_{y_i} to denote $\theta_{y_i|z} = \theta_{y_i|\bar{z}}$, and noting that $\mathcal{P}^\star(y) = P(y)$ in the selected BN, we get:

$$\begin{aligned}
\mathcal{P}^\star(y) &= \sum_{\mathbf{y}} \sum_{i=1}^{N} \mathcal{P}^\star(y \mid I{=}i, \mathbf{y}) \mathcal{P}^\star(I{=}i, \mathbf{y}) \\
&= \sum_{\mathbf{y}} \sum_{i=1}^{N} \mathcal{P}^\star(y \mid I{=}i, \mathbf{y}) \cdot \theta_{I=i} \cdot \prod_{i=1}^{N} \theta_{y_i} \\
&= \sum_{i=1}^{N} \theta_{I=i} \cdot \theta_{y_i} = \frac{1}{N} \sum_{i=1}^{N} \theta_{y_i}.
\end{aligned}$$

Intuitively, each node Y_i is either *activated* ($\theta_{y_i} = 1$) or *de-activated* ($\theta_{y_i} = 0$), where $\mathcal{P}^\star(y)$ is the proportion of activated nodes. Moreover, by choice of thresholds T_z and $T_{\bar{z}}$, as x increases, more nodes Y_i get activated. In fact, exactly $\lfloor N \cdot f(x) \rfloor$ are activated, leading to $\mathcal{P}^\star(y) = \frac{1}{N} \lfloor N \cdot f(x) \rfloor$ and $|f(x) - \mathcal{P}^\star(y)| \leq \varepsilon$. □

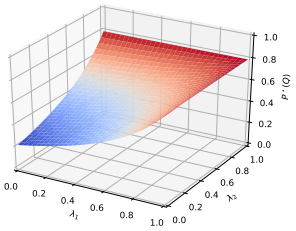
6. The Functional Form of TACs

Consider a TBN that contains some binary nodes, $E_1, \ldots, E_n, Q$. Soft evidence on nodes E_i corresponds to a vector $\lambda_1, \ldots, \lambda_n$ in $[0,1]^n$. Hence, a query that asks for the probability of $Q{=}q$ given soft evidence will then correspond to a function that maps $[0,1]^n$ into $[0,1]$. In Section 5 and Appendix B, we provided a universal approximation theorem, showing how each continuous function from $[0,1]^n$ into $[0,1]$ can be approximated to an arbitrary error by a TBN query. The construction used a TBN and query that are based on the given function and error. In practice though, the TBN and query are mandated by modeling and task considerations, which may restrict the class of functions that can be learned from data. We next show, however, that regardless of the TBN and query, the induced function must be *piecewise multi-linear*.

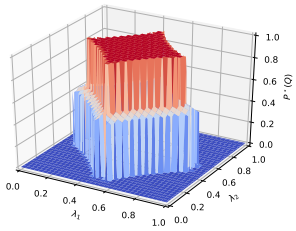
Definition 1. *We say that function $f(\lambda_1, \ldots, \lambda_n)$ is multi-linear if it has the form $\sum_{I \subseteq \{1,\ldots,n\}} C_I \prod_{i \in I} \lambda_i$ for some constants C_I. We say it is linear if it has the form $C_0 + \sum_{i=1}^{n} C_i \lambda_i$ for some constants C_i.*

For example, $f(\lambda_1, \lambda_2) = a\lambda_1\lambda_2 + b\lambda_1 + c\lambda_2 + d$ is multi-linear and $f(\lambda_1, \lambda_2) = a\lambda_1 + b\lambda_2 + c$ is linear.

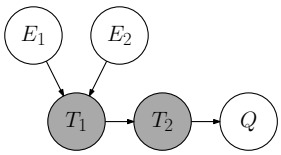
We will next illustrate the functional form of BN and TBN queries using concrete examples, then follow by the formal results. We will start by assuming only one evidence node E with soft evidence $\lambda \in [0,1]$. Consider now the following queries:



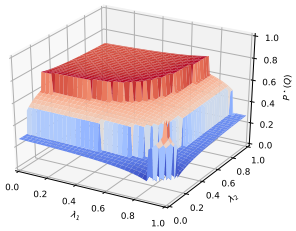
(a) BN function



(b) TBN function



(c) BN/TBN structure



(d) TBN function

Figure 8: Conditional probability functions, $F(\lambda_1, \lambda_2)$, induced by BN and TBN queries. The TBN functions are for different parameters and thresholds, where nodes T_1 and T_2 are testing.

— $f(\lambda)$: the *joint* probability of $Q{=}q$ and evidence λ, i.e., $P^\star(q)$.

— $F(\lambda)$: the *conditional* probability of $Q{=}q$ given evidence λ, i.e., $\mathcal{P}^\star(q)$.

Figure 7(a) depicts these functions for a BN $E \to T \to Q$ with some arbitrary parameters. Figure 7(b) depicts these functions for a TBN with the same structure, but where node T is testing (again, the parameters and thresholds are arbitrary). For the BN, $f(\lambda)$ is a linear function and $F(\lambda)$ is a quotient of two linear functions. For the TBN, $f(\lambda)$ is a piecewise linear function and $F(\lambda)$ is a piecewise quotient of two linear functions.

More precisely, for BNs with one evidence node, these functions have the following form (Castillo et al., 1996; Jensen, 1999; Kjærulff & van der Gaag, 2000; Darwiche, 2000):

$$f(\lambda) = a\lambda + b \qquad F(\lambda) = \frac{a\lambda + b}{c\lambda + d}$$

where the constants a, b, c, d depend only on the BN parameters. For TBNs with one evidence node, the input space $[0, 1]$ is partitioned into *regions,* where functions f and F take the above form but with different constants in each region; see Figure 7(b).

Suppose we now have two evidence nodes E_1 and E_2 with soft evidence λ_1 and λ_2. For TBNs, the input space $[0, 1] \times [0, 1]$ is partitioned into two-dimensional regions. The functional form in each region is:

$$f(\lambda_1, \lambda_2) = a\lambda_1\lambda_2 + b\lambda_1 + c\lambda_2 + d \qquad F(\lambda_1, \lambda_2) = \frac{a\lambda_1\lambda_2 + b\lambda_1 + c\lambda_2 + d}{e\lambda_1\lambda_2 + f\lambda_1 + g\lambda_2 + h}$$

Figure 8 depicts examples of $F(\lambda_1, \lambda_2)$ corresponding to BN and TBN queries (only one region for the BN).

We now have the following result, showing that BN queries compute multi-linear functions.

Theorem 2. *Consider a BN that contains some binary nodes $E_1, \ldots, E_n, Q$ and let $\lambda_1, \ldots, \lambda_n$ denote a soft evidence vector on nodes $E_1, \ldots, E_n$. There are constants C_I for $I \subseteq \{1, \ldots, n\}$ such that*

$$P^\star(q) = \sum_I C_I \prod_{i \in I} \lambda_i.$$

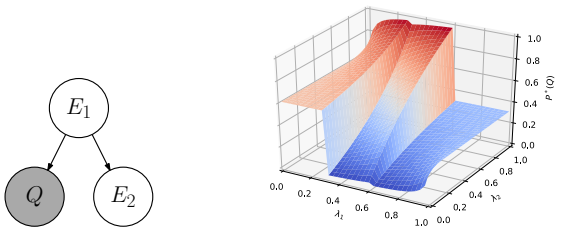


Figure 9: A conditional probability function, $F(\lambda_1, \lambda_2)$, induced by a TBN query. Node Q is testing.

Proof Consider an instantiation $\mathbf{x}$ over all network variables. We write $\mathbf{x} \models x\mathbf{u}$ to mean that instantiation $\mathbf{x}$ sets variable X and its parents $\mathbf{U}$ to x and $\mathbf{u}$. Let I^+ be the indices i of evidence variables E_i set positively by $\mathbf{x}$, and let I^- be the indices i of evidence variables E_i set negatively by $\mathbf{x}$. We have:

$$\begin{aligned} P^\star(\mathbf{x}) &= \prod_{\mathbf{x}\models x\mathbf{u}} \theta_{x|\mathbf{u}} \cdot \prod_{i\in I^+} \lambda_i \cdot \prod_{i\in I^-} (1-\lambda_i) \\ &= \prod_{\mathbf{x}\models x\mathbf{u}} \theta_{x|\mathbf{u}} \cdot \prod_{i\in I^+} \lambda_i \cdot \Big[\sum_{J\subseteq I^-} (-1)^{|J|} \cdot \prod_{j\in J} \lambda_j\Big] \\ &= \sum_{J\subseteq I^-} (-1)^{|J|} \cdot \prod_{\mathbf{x}\models x\mathbf{u}} \theta_{x|\mathbf{u}} \cdot \prod_{i\in I^+} \lambda_i \cdot \prod_{j\in J} \lambda_j \\ &= \sum_{I\subseteq\{1,\ldots,n\}} D_I^{\mathbf{x}} \prod_{i\in I} \lambda_i \end{aligned}$$

where $D_I^{\mathbf{x}}$ is either 0, $P(\mathbf{x}) = \prod_{\mathbf{x}\models x\mathbf{u}} \theta_{x|\mathbf{u}}$ or $-P(\mathbf{x})$. Hence, $P^\star(\mathbf{x})$ is a multi-linear function. Since $P^\star(q) = \sum_{\mathbf{x}\models q} P^\star(\mathbf{x})$, it follows that $P^\star(q)$ is also multi-linear as it is the sum of multi-linear functions. □

The following result shows that TBN queries compute piecewise multi-linear functions.

Theorem 3. *Consider a TBN that contains some binary nodes $E_1, \ldots, E_n, Q$ and let $\lambda_1, \ldots, \lambda_n$ denote a soft evidence vector on nodes $E_1, \ldots, E_n$. The space $[0,1]^n$ can be partitioned into a finite set of regions R that satisfy the following. For each region $r \in R$, there are constants $C_{I,r}$ for $I \subseteq \{1, \ldots, n\}$ such that*

$$P^\star(q) = \sum_I C_{I,r} \prod_{i\in I} \lambda_i, \text{ for vectors } \lambda_1, \ldots, \lambda_n \text{ in region } r.$$

Proof For a given soft evidence vector $\lambda_1, \ldots, \lambda_n$, the TBN selects a unique set of CPTs, leading to the selection of a unique BN. There is a finite number of possible selections that a TBN can make, leading to a finite number of BNs. Hence, the soft evidence space of $[0,1]^n$ can be partitioned into a finite set of regions R, with each region $r \in R$ leading to a unique BN. By Theorem 2, $P^\star(q)$ takes the form in Theorem 3. □

While TBN queries represent piecewise *multi-linear* functions, neural networks with ReLU activation functions represent piecewise *linear* functions (Pascanu et al., 2014; Montúfar et al., 2014). Moreover, there has been work on bounding the number of regions for such functions, depending on the size and depth of neural networks, e.g., Pascanu et al. (2014); Montúfar et al. (2014); Raghu et al. (2017); Serra et al. (2018).

A TBN query can induce different piecewise functions, depending on the TBN parameters and thresholds that are specified or learned from data (Figures 8(b) and 8(d) depict two functions induced by the same TBN query). In particular, these functions may have regions that differ in their nature and count, which impacts how well the learned function fits the training data.

Definition 2. *For a piecewise function, the number of regions is called the function granularity. For a TBN query, the query granularity is the maximum granularity attained by any function inducible by that query.*

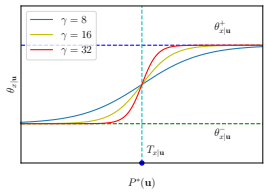


Figure 10: CPT selection using a sigmoid function.

The granularity of a TBN query—and, to an extent, its expressiveness—depends on the location of evidence, query, and testing nodes. For example, in Figure 9, evidence λ_2 on node E_2 does not impact CPT selection at testing node Q since E_2 is not an ancestor of Q. Hence, the regions are one-dimensional, across evidence λ_1. Contrast this with the TBN query in Figure 8(c). In this case, CPT selection is impacted by evidence on both E_1 and E_2, leading to two-dimensional regions as in Figures 8(b) and 8(d).

For completeness, we close this section by the following immediate result.

Theorem 4. *The function computed by a neural network with ReLU and step activation functions can be computed by a TAC with size proportional to the neural network size.*[13]

Proof A testing unit $f(x, T)$ with parameters $\theta^+ = 1$ and $\theta^- = 0$ corresponds to a step activation function with threshold T. Moreover, such a testing unit can emulate a ReLU $g(x)$ as follows: $g(x) = x \cdot f(x, 0) = \max(0, x)$. A neuron with a linear activation function can be simulated by an AC using multipliers and adders. A neuron with a ReLU or step activation functions can be simulated by a TAC using the above AC and the TAC fragment of the activation function. □

7. Generalized CPT Selection

TBNs get their expressiveness from the ability to select CPTs based on the available evidence. Selecting CPTs based on threshold tests, e.g., $\mathcal{P}^\star(\mathbf{u}) \geq T_{X|\mathbf{u}}$, is both simple and sufficient for universal approximation. However, one can employ more general and refined selection schemes, which can also facilitate the learning of TAC parameters and thresholds using gradient descent methods. For example, one can use a sigmoid function to select CPTs based on the following equations; see Figure 10:[14]

$$\begin{aligned} \theta_{x|\mathbf{u}} &= \tau_{\mathbf{u}} \cdot \theta^+_{x|\mathbf{u}} + (1 - \tau_{\mathbf{u}}) \cdot \theta^-_{x|\mathbf{u}} \\ \tau_{\mathbf{u}} &= [1 + \exp\{-\gamma \cdot (\mathcal{P}^\star(\mathbf{u}) - T_{X|\mathbf{u}})\}]^{-1} \end{aligned} \tag{1}$$

Here, γ is a meta-parameter that controls the sigmoid slope and $\tau_{\mathbf{u}} \in [0, 1]$. As γ tends to ∞, this selection scheme tends towards implementing a threshold test. Moreover, as γ tends to 0, the selection tends towards a fixed CPT as in BNs. Threshold tests select a CPT for node X from a finite set of 2^n different CPTs, where n is the number of states for parents $\mathbf{U}$. However, the sigmoid selects a weighted average of these CPTs, which is another CPT.[15] The TAC compilation algorithm in Appendix C can easily accommodate this more general selection scheme, leading to TACs with sigmoid units that replace testing units.

We next show some results on training TACs to approximate functions using labeled data and gradient descent. We considered the TBN in Figure 8(c), except that we used four testing nodes, $T_1, \ldots, T_4$. We

[13] The resulting TAC may have negative parameters and thresholds.

[14] In general, the selection of a CPT parameter $\theta_{x|\mathbf{u}}$ is a process that accepts as input the posterior $\mathcal{P}^\star(\mathbf{u})$, a threshold $T_{X|\mathbf{u}}$, and the two parameters $\theta^+_{x|\mathbf{u}}$ and $\theta^-_{x|\mathbf{u}}$ to be selected from. In the case of sigmoid selection, we use the sigmoid function to produce a selection $\theta_{x|\mathbf{u}}$ that is a weighted average of $\theta^+_{x|\mathbf{u}}$ and $\theta^-_{x|\mathbf{u}}$.

[15] If $P_1(X)$ and $P_2(X)$ are distributions, and if $\tau \in [0, 1]$, then $P(X) = \tau \cdot P_1(X) + (1 - \tau) \cdot P_2(X)$ is also a distribution.

Table 1: Functions $f_1, \ldots, f_5$ and their TAC and AC approximations.

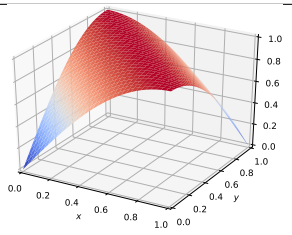
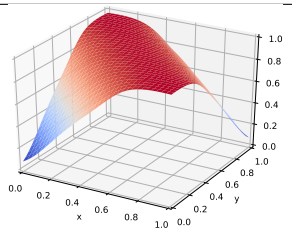
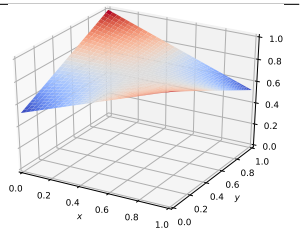
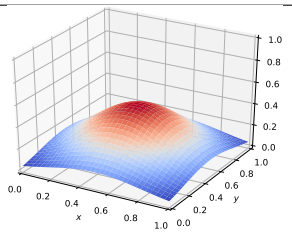
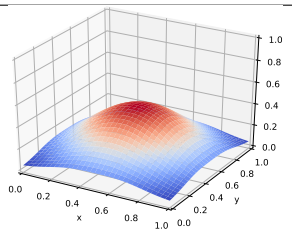
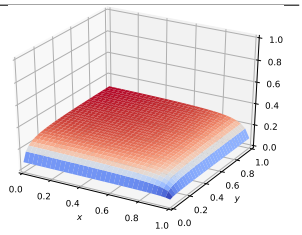
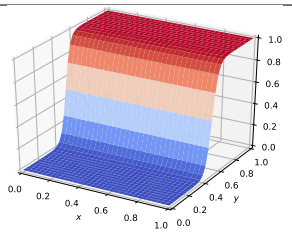
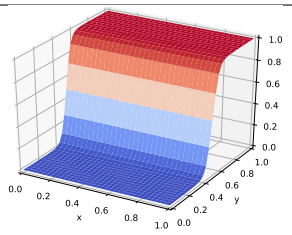
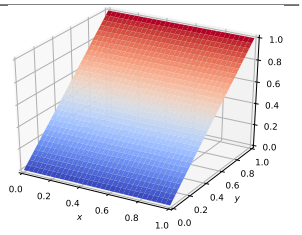
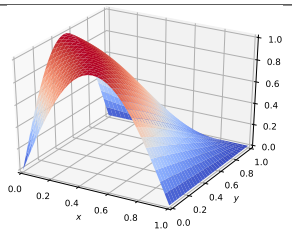
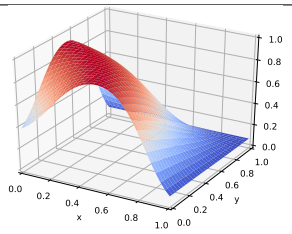
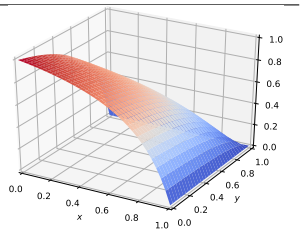
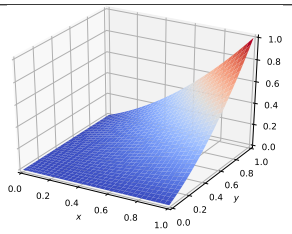
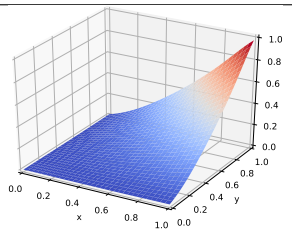
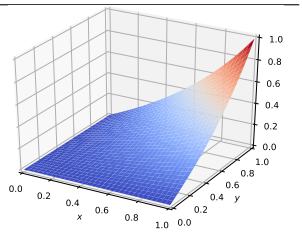
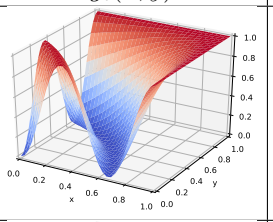
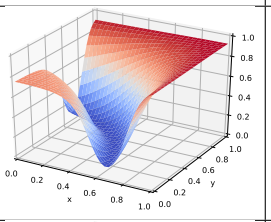
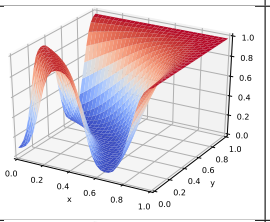
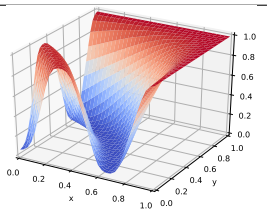
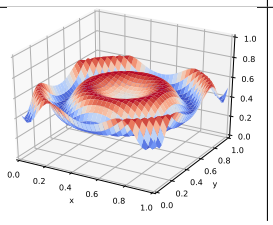
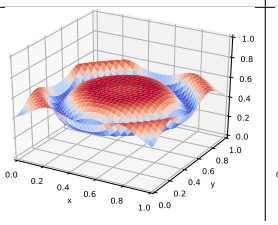
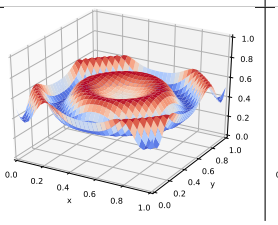
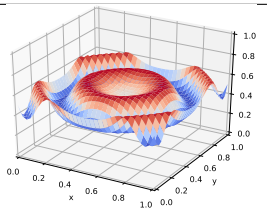
i	$f_i(x, y)$	learned TAC function	learned AC function
1			
2			
3			
4			
5			

Table 2: Functions g_1 and g_2 and their TAC approximations with increasing number of testing nodes.

i	$g_i(x, y)$	4-test TBN	8-test TBN	16-test TBN
1				
2				

compiled a TAC assuming E_1 and E_2 are evidence nodes and Q is the query node.[16] We then generated labeled data from the following continuous functions, which are visualized in the first column of Table 1.

$$f_1(x, y) = \sin\left(\tfrac{\pi}{2} \cdot (2 - x - y)\right)$$
$$f_2(x, y) = \tfrac{1}{2} \cdot e^{-5\cdot(x-\frac{1}{2})^2 - 5\cdot(y-\frac{1}{2})^2}$$
$$f_3(x, y) = \left(1 + e^{-32(y-\frac{1}{2})}\right)^{-1}$$
$$f_4(x, y) = \sin\left(\pi(1 - x)(1 - y)\right)$$
$$f_5(x, y) = \tfrac{1}{2} \cdot xy(x + y)$$

For each one of these functions, each mapping values from $[0, 1]^2$ to $[0, 1]$, we selected data examples (x, y) by dissecting the input space into a 32×32 grid and then recording the resulting values of $f(x, y)$ as labels. We then trained the TAC on this data using TensorFlow by minimizing mean-squared error as described in Appendix A.

The second column of Table 1 depicts the TAC approximations of these functions. The TAC approximates well all functions except for f_4, where the steep slope at the point $(x, y) = (0, 0)$ is not fit well. For comparison, we also depict the AC approximations of these functions in the third column of Table 1. Since ACs can only represent multi-linear functions or their quotients, we see that only function f_5 is approximated well. Moreover, the AC approximation of function f_4 is worse than the TAC approximation.

We now consider two additional functions that are visualized in the first column of Table 2:

$$g_1(x, y) = \tfrac{1}{2} + \tfrac{1}{2}\cos(3\pi(1 - x)(1 - y))$$
$$g_2(x, y) = \tfrac{1}{2} + \tfrac{1}{8}\sin(8\pi((x - \tfrac{1}{2})^2 + (y - \tfrac{1}{2})^2))$$

We approximated these functions using the same TBN query in Figure 8(c), but increased the number of testing nodes from 4 to 8 and finally to 16. As the number of testing nodes increased, so did the number of sigmoid units in the compiled TACs. The TAC approximations are depicted in the 2nd, 3rd and 4th columns of Table 2. As we increase the number of sigmoid units in the compiled TACs, we expect their expressiveness to also increase, leading to better approximations. This expectation is confirmed by Table 2.

8. Conclusion

We considered the relative expressiveness of Bayesian and neural networks. Neural networks are "universal approximators" of continuous functions, whereas marginal Bayesian network queries correspond to

[16] The TAC computes the conditional probability of $Q{=}q$ given soft evidence, $\mathcal{P}^\star(q)$.

multi-linear functions (joint queries) or their quotients (conditional queries). We proposed Testing Bayesian Networks (TBN) whose marginal queries are also "universal approximators," and Testing Arithmetic Circuits (TAC) for computing these queries. Moreover, we showed that marginal TBN queries and their TACs represent piecewise multi-linear functions, while highlighting that neural networks with ReLU activation functions represent piecewise linear functions. Finally, we generalized the concept of CPT selection that is responsible for the expressiveness of TBNs, which facilitated their training using gradient descent.

TBNs and TACs move us a step forward towards fusing model-based and function-based approaches to AI. In particular, they provide a framework for integrating background knowledge into expressive functions that can be learned from labeled data. This can contribute to learning more robust functions based on less data, and to verifying, interpreting and explaining learned functions.

Acknowledgments

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

Appendix A. Training TACs

We implemented a training algorithm for TACs using TensorFlow, which is based on a symbolic representation of the loss function to be minimized. In this tool, gradients are computed automatically and an optimizer (e.g., gradient descent method) can be used to minimize the given loss function. Using TensorFlow terminology, we assume training data with real-valued *features* in $[0, 1]$ and real-valued *labels* in $[0, 1]$.

TAC inputs assert soft evidence while the TAC output computes the conditional probability of some variable Q given soft evidence. The goal is to learn TAC parameters and thresholds that minimize the difference between TAC outputs and the labels of corresponding training examples. We used mean squared error as our loss function:

$$\frac{1}{N}\sum_{i=1}^{N}(TAC(\lambda_i) - y_i)^2.$$

We have N training examples, with the i-th example consisting of input vector λ_i and label y_i. $TAC(\lambda_i)$ is the TAC output under input vector λ_i. It is the result of normalizing two TAC nodes for $P^\star(q)$ and $P^\star(\bar{q})$. Since the parameters of our TAC are probabilities, we use a logistic function to represent them:

$$\theta_x = \frac{1}{1 + \exp\{-\tau_x\}} \qquad \theta_{\bar{x}} = \frac{\exp\{-\tau_x\}}{1 + \exp\{-\tau_x\}}$$

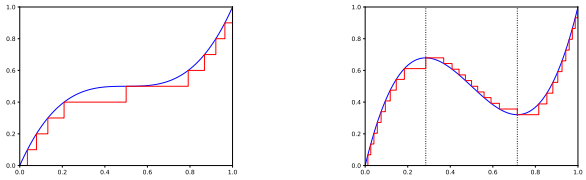
and optimize instead the real-valued meta-parameters τ_x. Our training algorithm assumed TACs with sigmoid selection units as given by Equation 1 (we used $\gamma = 8$).[17] To facilitate the training of longer chain-structured TBNs, we initialized the parameters using parameters learned from shorter chain-structured TBNs. Finally, while our description of the training algorithm is based on binary variables, the treatment applies directly to multi-valued variables too.

Appendix B. Universal Approximation Theorem

We next extend Theorem 1 of Section 5 to non-monotonic functions and then to multivariate functions.

Figure 11(b) depicts a non-monotonic function $f(x)$ from $[0, 1]$ to $[0, 1]$. To approximate $f(x)$ with a TBN, we first split our function into monotonic pieces as shown in the figure (we used vertical dotted lines to mark the points where the sign of the first derivative $\frac{df}{dx}$ changes).

[17] Testing units (see Section 4) are challenging for gradient descent methods as they are not differentiable at $x = T$ and have a zero gradient everywhere else (with respect to input x).



(a) monotonic function (b) non-monotonic function

Figure B.11: Two functions and their approximations.

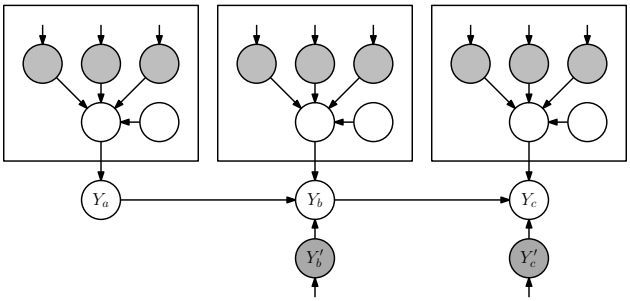


Figure B.12: A chain of TBNs.

Consider now the TBN of Figure B.12 that approximates this non-monotonic function using the TBN in Figure 6(a) for approximating monotonic functions as a building block (for clarity, in Figure B.12, we do not draw node Z on which soft evidence is asserted). Each of the three sub-networks enclosed in a box approximates each of the three monotonic components of our function $f(x)$. Let a, b and c denote the three components, and let T_{ab} and T_{bc} denote the values of x at the two borders. We have a chain of nodes $Y_a \rightarrow Y_b \rightarrow Y_c$ at the bottom of our network. Node Y_a simply copies the value of its parent (its CPT is an equivalence constraint). Node Y_b will either copy the value of Y_a or the value of its component b, depending on whether the input x is below or above the threshold T_{ab}. Node Y_c will either copy the value of Y_b or the value of its component c, depending on whether the input x is below are above the threshold T_{bc}. Testing nodes Y'_b and Y'_c perform these threshold tests and select CPTs that clamp themselves to a particular value, which determines where Y_b and Y_c copy their values from. This sequence of threshold tests will, given an input x, select the approximation of $f(x)$ from the appropriate component, which is finally the probability of Y_c. The size of this construction is linear in the number of times the sign of the first derivative changes.

The generalization to multivariate functions is analogous to the approximation of functions using "ridge" and "bump" functions; see, e.g., (Jones, 1990; Lapedes & Farber, 1987). First, we use our construction for approximating a univariate function as a building block to approximate a function $f(x_1, x_2)$ over two variables. In particular, we construct a TBN for N univariate functions $f_{x_2}(x_1) = f(x_1, x_2)$ for N values of x_2 from 0 to 1. As we did previously for approximating non-monotonic univariate functions by pieces, we construct a chain of these N TBNs and copy the output of the appropriate component based on the input value of x_2. To approximate a function $f(x_1, \ldots, x_n)$ over n variables, we construct N TBNs that approximate functions $f_n(x_1, \ldots, x_{n-1})$ over $n-1$ variables, and then perform a similar construction.

The error in the approximation can be improved arbitrarily by increasing N (under some assumptions, i.e., the change in f is bounded for small changes in the input). Moreover, this construction is exponential in the number of input variables n. Related constructions for showing neural networks (with one or two hidden layers) are "universal approximators" are also exponential in the dimension of the function.

Appendix C. Compiling TBN Queries into TACs

A BN query can be compiled into an AC; see Figure 3. Similarly, a TBN query can be compiled into a TAC; see Figure 5. Compiling an AC can be done by keeping a symbolic trace of an elimination algorithm as described in (Darwiche, 2009, Chapter 12). We next provide an algorithm for compiling a TAC by keeping a symbolic trace of the factor elimination algorithm described in (Darwiche, 2009, Chapter 7). The algorithm assumes TBNs with threshold-based selection, but can be easily adjusted to handle sigmoid-based selection.

Appendix C.1. Factor Elimination

A *factor* $f(\mathbf{X})$ is a mapping from instantiations $\mathbf{x}$ into positive, real numbers. Factor elimination is a variation on variable elimination (Zhang & Poole, 1996; Dechter, 1996), in which factors are systematically eliminated according to the following process. Consider a factor $f(\mathbf{X})$, where variables $\mathbf{Y} \subseteq \mathbf{X}$ appear only in this factor. To eliminate this factor, we simply sum-out variables $\mathbf{Y}$ from the factor, leading to $\sum_{\mathbf{Y}} f(\mathbf{X})$, and then multiply the result by some other factor.[18] We can use this method to compute the marginal on any query variable Q in a BN. We start with the network CPTs as our initial factors. We then identify a factor that contains Q and call it the *root factor*. We successively eliminate all factors, one by one, except for the root factor. At this point, we are left with a single factor $f(\mathbf{Z}, Q)$, where $g(Q) = \sum_{\mathbf{Z}} f(\mathbf{Z}, Q)$ is the marginal on Q. Any elimination order works, but the specific order used impacts the algorithm's complexity.

We can assert soft evidence $(\lambda_1, \ldots, \lambda_k)$ on a variable X by adjusting its CPT as follows. The entry of each row corresponding to value x_i of X is multiplied by λ_i. If we have evidence, the computed factor $g(Q)$ needs to be normalized to obtain the posterior distribution on variable Q.

Appendix C.2. Symbolic Factor Elimination

A TBN query can be compiled into a TAC by keeping a symbolic trace of the factor elimination algorithm. This requires working with *symbolic* factors, whose entries are circuit nodes instead of numbers, and is analogous to compiling a BN into an AC by keeping a symbolic trace of elimination algorithms (Chavira & Darwiche, 2007; Darwiche, 2009). We next illustrate these concepts by providing an example of compiling an AC for a BN query. We then show how the technique can be extended to compiling TACs, also using a concrete example. We then follow by a formal statement of the compilation algorithm and its complexity.

Consider a BN over binary variables A, B, C with edges $A \rightarrow B$ and $A \rightarrow C$ and the following CPTs.

A	Θ_A
a	θ_a
$\bar{a}$	$\theta_{\bar{a}}$

A	B	$\Theta_{B\|A}$
a	b	$\theta_{b\|a}$
a	$\bar{b}$	$\theta_{\bar{b}\|a}$
$\bar{a}$	b	$\theta_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$\theta_{\bar{b}\|\bar{a}}$

A	C	$\Theta_{C\|A}$
a	c	$\theta_{c\|a}$
a	$\bar{c}$	$\theta_{\bar{c}\|a}$
$\bar{a}$	c	$\theta_{c\|\bar{a}}$
$\bar{a}$	$\bar{c}$	$\theta_{\bar{c}\|\bar{a}}$

Suppose we have soft evidence on variables A and C and want to compile an AC that computes the marginal $(P^\star(b), P^\star(\bar{b}))$. We start with the following CPTs that incorporate evidence (we use $+(n_1, n_2)$ and $*(n_1, n_2)$ to denote sum and product circuit nodes with n_1 and n_2 as their children).

A	Θ_A
a	$n_1 = *(\lambda_a, \theta_a)$
$\bar{a}$	$n_2 = *(\lambda_{\bar{a}}, \theta_{\bar{a}})$

A	B	$\Theta_{B\|A}$
a	b	$\theta_{b\|a}$
a	$\bar{b}$	$\theta_{\bar{b}\|a}$
$\bar{a}$	b	$\theta_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$\theta_{\bar{b}\|\bar{a}}$

A	C	$\Theta_{C\|A}$
a	c	$n_3 = *(\lambda_c, \theta_{c\|a})$
a	$\bar{c}$	$n_4 = *(\lambda_{\bar{c}}, \theta_{\bar{c}\|a})$
$\bar{a}$	c	$n_5 = *(\lambda_c, \theta_{c\|\bar{a}})$
$\bar{a}$	$\bar{c}$	$n_6 = *(\lambda_{\bar{c}}, \theta_{\bar{c}\|\bar{a}})$

Our root factor is $\Theta_{B|A}$. We will eliminate factor $\Theta_{C|A}$ first. Summing out variable C from the factor gives

A	$\sum_C \Theta_{C\|A}$
a	$n_7 = +(n_3, n_4)$
$\bar{a}$	$n_8 = +(n_5, n_6)$

[18] Summing-out and multiplication are standard operations on factors; see, e.g., (Darwiche, 2009, Chapter 6).

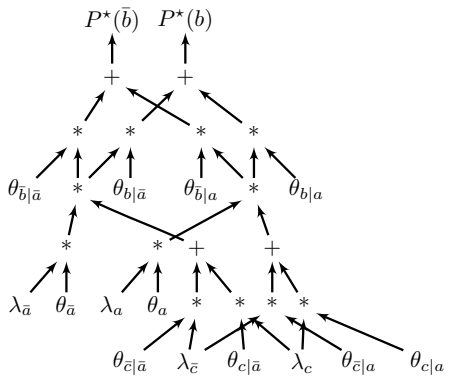


Figure C.13: An AC compiled from a BN query.

Multiplying the result by factor Θ_A, we get

A	$\Theta_A \sum_C \Theta_{C\|A}$
a	$n_9 = *(n_1, n_7)$
$\bar{a}$	$n_{10} = *(n_2, n_8)$

Our set of factors are now

A	$\Theta_A \sum_C \Theta_{C\|A}$
a	$n_9 = *(n_1, n_7)$
$\bar{a}$	$n_{10} = *(n_2, n_8)$

A	B	$\Theta_{B\|A}$
a	b	$\theta_{b\|a}$
a	$\bar{b}$	$\theta_{\bar{b}\|a}$
$\bar{a}$	b	$\theta_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$\theta_{\bar{b}\|\bar{a}}$

Eliminating the recently constructed factor, we now have

A	B	$\Theta_{B\|A}\Theta_A \sum_C \Theta_{C\|A}$
a	b	$n_{11} = *(n_9, \theta_{b\|a})$
a	$\bar{b}$	$n_{12} = *(n_9, \theta_{\bar{b}\|a})$
$\bar{a}$	b	$n_{13} = *(n_{10}, \theta_{b\|\bar{a}})$
$\bar{a}$	$\bar{b}$	$n_{14} = *(n_{10}, \theta_{\bar{b}\|\bar{a}})$

Summing out variable A gives

B	$\sum_A \Theta_{B\|A}\Theta_A \sum_C \Theta_{C\|A}$
b	$n_{15} = +(n_{11}, n_{13})$
$\bar{b}$	$n_{16} = +(n_{12}, n_{14})$

We have now compiled a circuit whose outputs, n_{15} and n_{16}, give the marginal for variable B; see Figure C.13. The inputs to this circuits are the vectors $(\lambda_a, \lambda_{\bar{a}})$ and $(\lambda_c, \lambda_{\bar{c}})$ representing soft evidence, in addition to network parameters. Dividing nodes n_{15} and n_{16} by their sum yields the posterior $(\mathcal{P}^\star(b), \mathcal{P}^\star(\bar{b}))$.

One can prune the Bayesian network before compiling an AC. In particular, one can successively remove leaf variables as long as they are not query or evidence variables.

Appendix C.3. Compiling TACs via Symbolic Factor Elimination

Beyond sum and product nodes, TACs also use *testing nodes* $[p \geq t \,?\, n^+ : n^-]$, which pass through the value of n^+ if $p \geq t$ and the value of n^- otherwise (like a multiplexer).

Compiling TACs is similar to compiling ACs but requires two phases. In the first phase, we *select* CPTs for all testing nodes that are relevant to the query. In the second phase, we perform classical *inference* as in the previous example. The selection phase converts testing CPTs into regular CPTs using an operation

called *flattening.* The flattening of a CPT for variable X involves only its ancestral CPTs. In particular, before the CPT for variable X is flattened, all its ancestral CPTs must be flattened and eliminated.

Consider our earlier example over binary variables A, B, C with edges $A \rightarrow B$ and $A \rightarrow C$, and suppose now that variable B is testing. We have soft evidence on variables A and C and want to compile a TAC that computes the marginal $(P^\star(b), P^\star(\bar{b}))$.

Selection Phase As we want to flatten the CPT for variable B, the CPT for variable C is irrelevant, so we start with the following CPTs.

A	Θ_A
a	$n_1 = *(\lambda_a, \theta_a)$
$\bar{a}$	$n_2 = *(\lambda_{\bar{a}}, \theta_{\bar{a}})$

A	B	$\Theta^?_{B\|A}$		
a	b	$T_{B\|a}$	$\theta^+_{b\|a}$	$\theta^-_{b\|a}$
a	$\bar{b}$	$T_{B\|a}$	$\theta^+_{\bar{b}\|a}$	$\theta^-_{\bar{b}\|a}$
$\bar{a}$	b	$T_{B\|\bar{a}}$	$\theta^+_{b\|\bar{a}}$	$\theta^-_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	$T_{B\|\bar{a}}$	$\theta^+_{\bar{b}\|\bar{a}}$	$\theta^-_{\bar{b}\|\bar{a}}$

The CPT for variable A is regular so it is notated as usual. The CPT for variable B is testing so it is notated differently. The second column is empty initially but will contain TAC nodes when factors are multiplied into the CPT. The third column contains thresholds and dynamic parameters and is kept until the CPT is flattened (i.e., converted into a regular CPT).

We first eliminate the CPT for variable A. No variables are summed out in this case, so we just multiply this CPT by the testing CPT for variable B, leading to

A	B	$\Theta_A\Theta^?_{B\|A}$			
a	b	n_1	$T_{B\|a}$	$\theta^+_{b\|a}$	$\theta^-_{b\|a}$
a	$\bar{b}$	n_1	$T_{B\|a}$	$\theta^+_{\bar{b}\|a}$	$\theta^-_{\bar{b}\|a}$
$\bar{a}$	b	n_2	$T_{B\|\bar{a}}$	$\theta^+_{b\|\bar{a}}$	$\theta^-_{b\|\bar{a}}$
$\bar{a}$	$\bar{b}$	n_2	$T_{B\|\bar{a}}$	$\theta^+_{\bar{b}\|\bar{a}}$	$\theta^-_{\bar{b}\|\bar{a}}$

If we define a new TAC node $n_3 = +(n_1, n_2)$, then n_1/n_3 and n_2/n_3 represent the posterior on variable A, which is what we need to flatten the CPT for variable B as follows[19]

A	B	$\Theta_{B\|A}$
a	b	$n_4 = [n_1 \geq *(T_{B\|a}, n_3)\ ?\ \theta^+_{b\|a} : \theta^-_{b\|a}]$
a	$\bar{b}$	$n_5 = [n_1 \geq *(T_{B\|a}, n_3)\ ?\ \theta^+_{\bar{b}\|a} : \theta^-_{\bar{b}\|a}]$
$\bar{a}$	b	$n_6 = [n_2 \geq *(T_{B\|\bar{a}}, n_3)\ ?\ \theta^+_{b\|\bar{a}} : \theta^-_{b\|\bar{a}}]$
$\bar{a}$	$\bar{b}$	$n_7 = [n_2 \geq *(T_{B\|\bar{a}}, n_3)\ ?\ \theta^+_{\bar{b}\|\bar{a}} : \theta^-_{\bar{b}\|\bar{a}}]$

This is a regular CPT since each entry is a circuit node. This finishes the selection phase, leading to the partial TAC in Figure 14(a). We are now ready for the inference phase.

Inference Phase Our CPTs are now all regular

A	Θ_A
a	n_1
$\bar{a}$	n_2

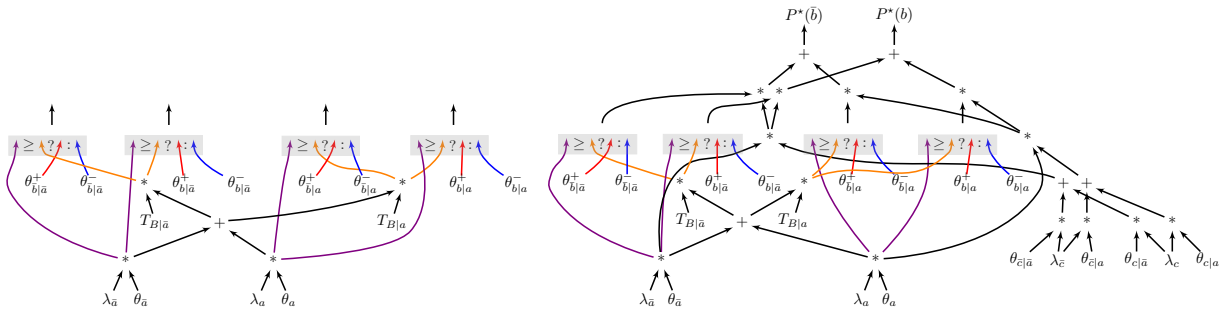
A	B	$\Theta_{B\|A}$
a	b	n_4
a	$\bar{b}$	n_5
$\bar{a}$	b	n_6
$\bar{a}$	$\bar{b}$	n_7

A	C	$\Theta_{C\|A}$
a	c	n_8
a	$\bar{c}$	n_9
$\bar{a}$	c	n_{10}
$\bar{a}$	$\bar{c}$	n_{11}

Inference can proceed as in the previous section: eliminate factor $\Theta_{C|A}$ then factor Θ_A. To eliminate factor $\Theta_{C|A}$, we sum-out variable C from the factor leading to

A	$\sum_C \Theta_{C\|A}$
a	$n_{12} = +(n_8, n_9)$
$\bar{a}$	$n_{13} = +(n_{10}, n_{11})$

[19] More generally: after flattening and eliminating all ancestral CPTs of a variable X, the CPT for X will contain the marginal on its parents given evidence on its ancestors. This marginal is all we needed to select a CPT for variable X, whether we are using threshold-based or sigmoid-based selection.



(a) Selection (b) Inference

Figure C.14: A TAC compiled from a TBN query.

We then multiply the above factor by factor Θ_A. Our new set of factors is

A	$\Theta_A \sum_C \Theta_{C\|A}$
a	$n_{14} = *(n_{12}, n_1)$
$\bar{a}$	$n_{15} = *(n_{13}, n_2)$

A	B	$\Theta_{B\|A}$
a	b	n_4
a	$\bar{b}$	n_5
$\bar{a}$	b	n_6
$\bar{a}$	$\bar{b}$	n_7

To eliminate the last constructed factor, we just multiply it by the factor for variable B, leading to

A	B	$\Theta_{B\|A}\Theta_A \sum_C \Theta_{C\|A}$
a	b	$n_{16} = *(n_{14}, n_4)$
a	$\bar{b}$	$n_{17} = *(n_{14}, n_5)$
$\bar{a}$	b	$n_{18} = *(n_{15}, n_6)$
$\bar{a}$	$\bar{b}$	$n_{19} = *(n_{15}, n_7)$

Summing-out variable A, we now have the final factor

B	$\sum_A \Theta_{B\|A}\Theta_A \sum_C \Theta_{C\|A}$
b	$n_{20} = +(n_{16}, n_{18})$
$\bar{b}$	$n_{21} = +(n_{17}, n_{19})$

We now have a TAC whose two outputs, n_{20} and n_{21}, compute the marginal $(P^\star(b), P^\star(\bar{b}))$; see Figure 14(b). Normalizing the circuit outputs gives the posterior $(\mathcal{P}^\star(b), \mathcal{P}^\star(\bar{b}))$.

Appendix C.4. The Compilation Algorithm

We now provide a more formal description of the compilation algorithm.

Testing CPTs A testing CPT for node X with parents $\mathbf{U}$ will initially have rows of the following form:

$\mathbf{u}\, x$		$T_{X\|\mathbf{u}}$	$\theta^+_{x\|\mathbf{u}}$	$\theta^-_{x\|\mathbf{u}}$

When factors are multiplied into this testing CPT, its rows will have the following form

$\mathbf{u}\, x$	n	$T_{X\|\mathbf{u}}$	$\theta^+_{x\|\mathbf{u}}$	$\theta^-_{x\|\mathbf{u}}$

where n is a TAC node. When the CPT is flattened (see below), its rows will have the following form

$\mathbf{u}\, x$	$[n \geq *(m, T_{X\|\mathbf{u}})\, ?\, \theta^+_{x\|\mathbf{u}} : \theta^-_{x\|\mathbf{u}}]$

where n and m are TAC nodes. It is now a regular CPT since the entry in each row is a testing TAC node.

Pseudocode To compile a TBN query into a TAC (Q is the query node):

1. *Pruning.* Repeatedly remove a leaf TBN node if it is not an evidence or query node.
2. *Initialization.* Replace each TBN parameter and threshold by a corresponding TAC node.
3. *Entering Evidence.* If a node X has soft evidence and k values, construct TAC nodes for $\lambda_1, \ldots, \lambda_k$ and multiply each row in the CPT of X by λ_i if the row corresponds to value x_i.
4. *Selection Phase.* Visit testing nodes X, ancestors before descendants.
   (a) Let Σ be the set of CPTs for X and its ancestors (ancestral CPTs must be regular at this point).
   (b) Eliminate all factors from Σ except the CPT for node X.
   (c) Flatten the CPT for node X with parents $\mathbf{U}$ as follows:
      i. Create a TAC node $n = \sum_{\mathbf{u}} n_{\mathbf{u}}$, where $n_{\mathbf{u}}$ is the TAC node in some CPT row for state $\mathbf{u}$.[20]
      ii. Replace each CPT row

$\mathbf{u}\, x$	$n_{\mathbf{u}}$	$T_{X\|\mathbf{u}}$ $\quad \theta^{+}_{x\|\mathbf{u}}$ $\quad \theta^{-}_{x\|\mathbf{u}}$

      by

$\mathbf{u}\, x$	$[n_{\mathbf{u}} \geq *(n, T_{X\|\mathbf{u}})\ ?\ \theta^{+}_{x\|\mathbf{u}} : \theta^{-}_{x\|\mathbf{u}}]$

   (the CPT for node X is now regular).
5. *Inference Phase.*
   (a) Let Σ be the set of all TBN CPTs (whether initially regular or flattened).
   (b) Let f be a factor in Σ that includes query node Q.
   (c) Eliminate all factors in Σ except for factor f.
   (d) Return $\sum_{\mathbf{Z}} f$, where $\mathbf{Z}$ are the variables of factor f except Q.

Entries of the returned factor will contain TAC outputs, which compute the marginal $P^\star(Q)$. Normalizing these outputs gives the posterior $\mathcal{P}^\star(Q)$.

The Order of Eliminating Factors The factor elimination algorithm is an abstraction of the jointree algorithm (Darwiche, 2009, Chapter 7). In particular, a message sent by the jointree algorithm from cluster i to cluster j can be interpreted as eliminating the factor at cluster i, leading to its multiplication by the factor at cluster j. Hence, the order in which factors are eliminated can be set using a jointree, which also determines the complexity of factor elimination. To eliminate a set of factors Σ except for some factor $f \in \Sigma$, we first assign each factor $g \in \Sigma$ to some jointree cluster that contains the variables of factor g. If multiple factors are assigned to the same cluster, we view this as one factor corresponding to their product. We next designate the cluster r containing factor f as the root cluster. We finally eliminate factors as follows. We only eliminate a factor if it is attached to a leaf cluster $i \neq r$, multiplying it into its neighboring factor (cluster i is also removed). The process terminates when we are left with the root cluster r. Using a binary jointree with n clusters and width w, the time and space complexity of factor elimination is $O(n \exp(w))$.[21]

Complexity The elimination of factors in Steps 4b and 5c can be implemented using the same jointree, constructed for the TBN produced by Step 1. Given a binary jointree with n clusters and width w, the time and space complexity of compiling a TAC (the selection and inference phases) is then $O(m \cdot n \exp(w))$, where m is the number of testing nodes. This can be improved if one uses a different jointree for each iteration of the selection phase since each of these iterations involves a subset of the TBN that may be less connected.

Unique Nodes The described algorithm may construct equivalent TAC nodes. To avoid this, we maintain a *unique-node table,* which stores each constructed TAC node with a key based on its type and its children's identities. Before a new node is constructed, the table is checked if an equivalent node was constructed earlier. If the test succeeds, the node is used. Otherwise, a new node is constructed and added to the table.

Sigmoid Selection The algorithm can be slightly adjusted to compile TBNs that employ other CPT selection mechanisms, as long as the selection process requires only the posterior on parents (sigmoid selection satisfies this property). We just adjust the operation of CPT flattening to construct and add the desired selection nodes instead of testing nodes.

[20] CPT rows for the same instantiation $\mathbf{u}$ of parents $\mathbf{U}$ must have the same TAC node (represents the marginal on state $\mathbf{u}$).

[21] A jointree is *binary* if each cluster has at most three neighbors. The *width* of a jointree is the size of its largest cluster -1. See (Darwiche, 2009, Chapter 7) for details.

References

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS)* (pp. 153–160).

Castillo, E. F., Gutiérrez, J. M., & Hadi, A. S. (1996). Goal oriented symbolic propagation in bayesian networks. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)* (pp. 1263–1268).

Chan, H., & Darwiche, A. (2005). On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, *163*, 67–90.

Chavira, M., & Darwiche, A. (2007). Compiling Bayesian networks using variable elimination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2443–2449).

Choi, A., & Darwiche, A. (2017). On relaxing determinism in arithmetic circuits. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML)* (pp. 825–833).

Choi, A., & Darwiche, A. (2018). On the relative expressiveness of Bayesian and neural networks. In *Proceedings of the 9th International Conference on Probabilistic Graphical Models (PGM)*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303–314.

Darwiche, A. (2000). A differential approach to inference in bayesian networks. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)* (pp. 123–132).

Darwiche, A. (2003). A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, *50*, 280–305.

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A. (2018). Human-level intelligence or animal-like abilities? *Communications of the ACM (CACM)*, *61*, 56–67.

Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 211–219).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Halpern, J. (1990). An analysis of first-order logics of probability. *Artificial Intelligence*, *46*, 311–350.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hornik, K., Stinchcombe, M. B., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359–366.

Jensen, F. V. (1999). Gradient descent training of Bayesian networks. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU)* (pp. 190–200).

Jones, L. K. (1990). Constructive approximations for neural networks by sigmoidal functions. *Proceedings of the IEEE*, *78*, 1586–1589.

Kisa, D., Van den Broeck, G., Choi, A., & Darwiche, A. (2014). Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Kjærulff, U., & van der Gaag, L. C. (2000). Making sensitivity analysis computationally efficient. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)* (pp. 317–325).

Lapedes, A. S., & Farber, R. M. (1987). How neural nets work. In *Neural Information Processing Systems (NIPS)* (pp. 442–456).

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*, 861–867.

McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*, 115–133.

Montúfar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS)* (pp. 2924–2932).

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, *56*, 71–113.

Nilsson, N. (1986). Probabilistic logic. *Artificial intelligence*, *28*, 71–87.

Pascanu, R., Montúfar, G., & Bengio, Y. (2014). On the number of inference regions of deep feed forward networks with piece-wise linear activations.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. MK.

Poole, D. (2003). First-order probabilistic inference. In *IJCAI* (pp. 985–991).

Raghu, M., Poole, B., Kleinberg, J. M., Ganguli, S., & Sohl-Dickstein, J. (2017). On the expressive power of deep neural networks. In *ICML*.

Ranzato, M., Poultney, C. S., Chopra, S., & LeCun, Y. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19 (NIPS)* (pp. 1137–1144).

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–408.

Saul, L. K., Jaakkola, T. S., & Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research (JAIR)*, *4*, 61–76.

Serra, T., Tjandraatmadja, C., & Ramalingam, S. (2018). Bounding and counting linear regions of deep neural networks. In *ICML*.

Shen, Y., Choi, A., & Darwiche, A. (2018). Conditional PSDDs: Modeling and learning with modular knowledge. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*.

Shen, Y., Huang, H., Choi, A., & Darwiche, A. (2019). Conditional independence in testing bayesian networks. In *Proceedings of the Thirty-Sixth International Conference on Machine Learning (ICML)* (pp. 5701–5709).

Vomlel, J. (2006). Noisy-or classifier. *Int. J. Intell. Syst.*, *21*, 381–398.

Zhang, N. L., & Poole, D. (1996). Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, *5*, 301–328.