Supervised Learning with Background Knowledge

Yizuo Chen and Arthur Choi and Adnan Darwiche

University of California, Los Angeles CA 90024, USA
yizuo.chen@ucla.edu, aychoi@cs.ucla.edu, darwiche@cs.ucla.edu

Abstract. We consider the task of supervised learning while focusing on the impact that background knowledge may have on the accuracy and robustness of learned classifiers. We consider three types of background knowledge: causal domain knowledge, functional dependencies and logical constraints. Our findings are set in the context of an empirical study that compares two classes of classifiers: Arithmetic Circuit (AC) classifiers compiled from Bayesian network models with varying degrees of background knowledge, and Convolutional Neural Network (CNN) classifiers. We report on the accuracy and robustness of such classifiers on two tasks concerned with recognizing synthesized shapes in noisy images. We show that classifiers that encode background knowledge need much less data to attain certain accuracies and are more robust against noise level in the data and also against mismatches between noise patterns in the training and testing data.

Keywords: Supervised learning; Bayesian networks, neural networks, arithmetic circuits; background knowledge.

1 Introduction

Supervised learning has become very influential recently and stands behind most real-world applications of AI. In supervised learning, one learns a *function* from labeled data, a practice that is now dominated by the use of neural networks; see [12, 13, 1, 23]. Supervised learning can be applied in other contexts as well, such as causal models in the form of Bayesian networks [19, 20, 21, 26, 11]. Given a Bayesian network, one can *compile* an Arithmetic Circuit (AC) that maps evidence (inputs) to the posterior distribution on labels of interest (output) [6, 14, 25, 2, 17, 27, 3, 8]. AC parameters, which correspond to Bayesian network parameters, can then be learned from labeled data using gradient descent [24, 6, 16]. Hence, like a neural network, the AC is a circuit that computes a function whose parameters can be learned from labeled data.

The use of ACs in this fashion can be viewed as *model-based* supervised learning, in contrast to *model-free* supervised learning using neural networks. Model-based supervised learning is attractive since it integrates *background knowledge,* which has a number of advantages: a reduced reliance on data, improved robustness and the ability to provide data-independent guarantees on the learned function. Despite these promises, this type of model-based supervised learning is not very common today and, hence, the underlying promises rarely exhibit concretely through empirical studies or real-world applications. There are a number of reasons for the missed opportunities. Some are profound and pertain to the fact that circuits compiled from causal models may not be expressive enough to capture the data-generating function when the used causal model is incomplete (e.g., missing nodes or edges) [28, 29, 15, 10, 9]. While a proposal has been extended recently to address this barrier [5, 4], two major barriers remain even when the causal model is complete. The first barrier pertains to the complexity of compiling ACs, which is PP-hard and can be quite challenging even for causal models of moderate size. The second barrier relates to the complexity of training circuits from large datasets [30]. This barrier is shared with neural networks, except that significant advances have been made on that front. Such advances have been applied to the training of ACs only very recently; see, e.g., [18, 22, 8]. This particularly includes the use of tensor graphs which is standard for neural networks and can lead to orders of magnitude savings in training time.

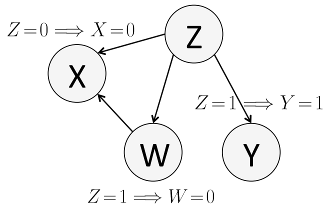
Motivated by recent advances on compiling and training ACs [8], we perform an empirical study in which we compare classifiers based on neural networks with ones based on ACs compiled from causal models.[1] We consider tasks for recognizing synthesized shapes in noisy images, and causal models that include different types of background knowledge: independence, functional dependencies, and logical constraints. We show that classifiers which integrate background knowledge need much less data to attain certain accuracies and are more robust against noise level in the data and also against mismatches between noise patterns in the training and testing data. While these results may appear intuitive and expected, the underlying experiments that support these conclusions were only possible due to the recent advances we mentioned earlier.

We start with some background material in Section 2 and follow by a description of the considered classification tasks in Section 3. The Bayesian network models we use for these tasks and corresponding AC classifiers are discussed in Section 4, with neural network classifiers discussed in Section 5. We then discuss data generation and training in Section 6, followed by experimental results in Sections 7 and 8. Motivated by the promise and relative scale of the reported experiments, we conclude by laying out in Section 9 a road map of further developments that are needed to advance the promises of model-based supervised learning so it reaches a level of practicality that model-free supervised learning has attained recently.

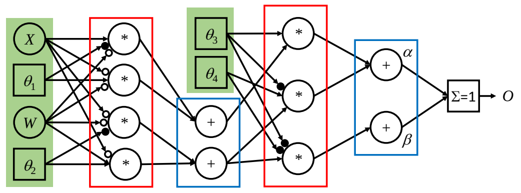
2 Background

A *Bayesian Network (BN)* is specified by a directed acyclic graph (DAG) and a set of *Conditional Probability Tables (CPTs).* The CPT for variable X with parents $\mathbf{U}$ specifies conditional probabilities $\theta_{x|\mathbf{u}} = Pr(x|\mathbf{u})$, known as network

[1] Some preliminary results are reported in [8], which had a narrower scope that did not include neural networks.


$Z=0 \Longrightarrow X=0$
$Z=1 \Longrightarrow Y=1$
$Z=1 \Longrightarrow W=0$
Z
X
W
Y

(a) BN with logical constraints (0/1 parameters)



(b) AC that computes query $O = Pr(y|x, w)$. Here, $\alpha = Pr(y, x, w)$ and $\beta = Pr(\bar{y}, x, w)$

Fig. 1. The AC uses adders (+), multipliers (∗), inverters (○), $1-\theta$ units (●), and normalizing units ($\Sigma = 1$). Excluding the Σ unit (division), which is not strictly needed for classification, we can emulate ○ and ● units using adders. The AC parameters $\theta_1, \ldots, \theta_4$ correspond to Bayesian network parameters (the AC implicitly integrates the 0/1 parameters of the Bayesian network).

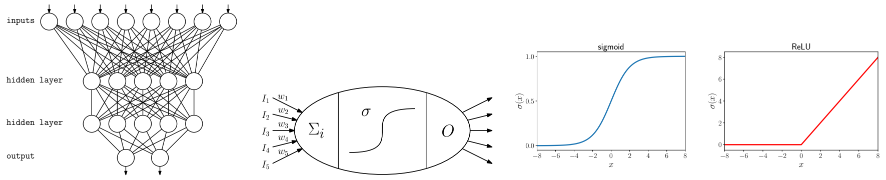


Fig. 2. A neural network structure, a neuron, and two activation functions.

parameters. If parameter $\theta_{x|\mathbf{u}} = 0$, the CPT contains a *logical constraint* (instantiation $x\mathbf{u}$ is impossible). If all CPT parameters are in $\{0, 1\}$, the CPT specifies a *functional dependency* (the state of X is determined by the state of $\mathbf{U}$).

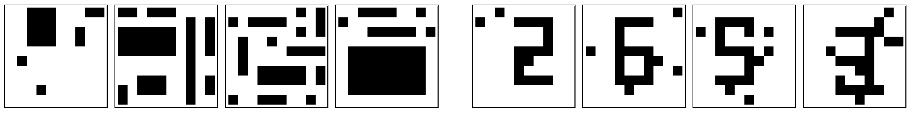
We consider Bayesian network and neural network classifiers in this paper. A *classifier* is a function c that maps a feature vector $\mathbf{x}$ to a class label y. A Bayesian network classifier is induced by designating some network variables $\mathbf{X}$ as features and some network variable Y as the class. The corresponding classification function typically has the form: $c(\mathbf{x}) = \text{argmax}_y\, Pr(y|\mathbf{x})$, which selects a class label that attains the highest conditional probability. A neural network classifier, in contrast, tries to approximate the function $c(\mathbf{x})$ directly by *fitting* a function to data.

The classification function of a Bayesian network can be represented using an AC (see Figure 1), which can be compiled from the Bayesian network; see, e.g., [6, 2, 8]. The classification function of a neural network can also be represented using a circuit with the addition of activation nodes. In particular, a neural network is composed of neurons, where each neuron computes $\sigma(\sum_i w_i x_i)$. Here, w_i is called a neuron's weight and σ is called an activation function; see Figure 2.

Both approaches can learn their classifier parameters from data. In a Bayesian network, the learned parameters are *conditional probabilities* that populate the network CPTs. In a neural network, the learned parameters are the *weights* of the neurons. A Bayesian network typically represents a *model* that can encode certain types of *background knowledge* such as cause-and-effect relationships, functional dependencies between variables, and logical constraints. Moreover, when compiling a Bayesian network into an AC, any background knowledge encoded in the network carries over to the AC as well; see [7]. In contrast, the neural network approach assumes that the most salient relationships between the features and the label can be learned from the data.

3 Classification Tasks: Recognizing Shapes

We consider two classification tasks, for rectangles (left) and digits (right).



We first consider the task of detecting rectangles in 10×10 black-and-white noisy images. A rectangle is characterized by the row and col of its upper-left corner, its width and height. A rectangle can have a shape, which is either tall (height $>$ width) or wide (height $<$ width). Variables row and col have integer values from 1 to 10 corresponding to the coordinate of the upper-left corner. Variables width and height also have integer values from 1 to 10. We train classifiers that predict rectangle properties in noisy images. Each classifier has 100 inputs corresponding to the image pixels, and outputs for shape, row, col, width and height.

Our second task is to classify a seven-segment digit in a 10×10 black-and-white noisy image. A seven-segment digit is composed of a selection of three horizontal segments and four vertical segments. For example, digit 8 is generated by selecting all seven segments and digit 0 is generated by selecting all segments but the middle horizontal segment. We want to train a classifier that predicts which digit is rendered in a noisy image. Each classifier has 100 inputs corresponding to the image pixels, and one output for each digit from 0 to 9.

4 AC Classifiers from Bayesian Network Models

Our first classifiers are based on generative models (Bayesian networks).

Rectangle Model We consider the generative model of rectangles shown in Figure 3.[2] The value of height is limited by the value of row, and the value of width is limited by the value of col. Variable row_i indicates whether the rectangle will render in row i so it is determined by the values of row and height. Similarly, each col_j is determined by the values of col and width. Finally, each variable $\mathsf{pixel}_\mathsf{ij}$ is determined by row_i and col_j where $\mathsf{pixel}_\mathsf{ij} = \mathsf{on}$ iff $\mathsf{row}_\mathsf{i} = \mathsf{on}$ and $\mathsf{col}_\mathsf{j} = \mathsf{on}$. However, to generate noisy rectangle images, we would need a distribution for

[2] Haiying Huang proposed this particular rectangle model.

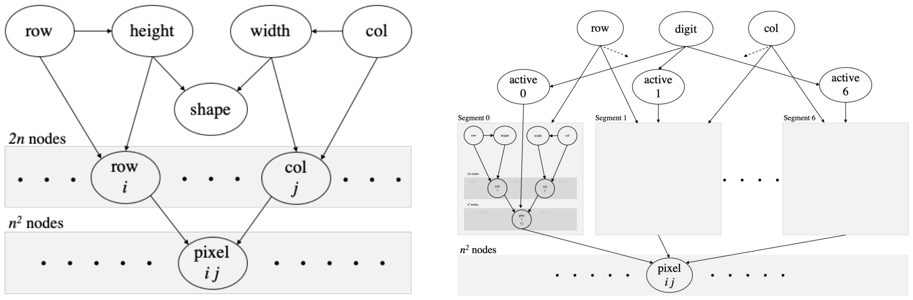


Fig. 3. Generative models for rectangles (top) and seven-segment digits (bottom).

each variable pixel_{ij} that is conditioned on the values of row_i and col_j, where such distributions are learned from data. The other distributions that get learned from data are the prior distributions for row and col (upper-left corner of the rectangle) in addition to the conditional distributions for height and width.

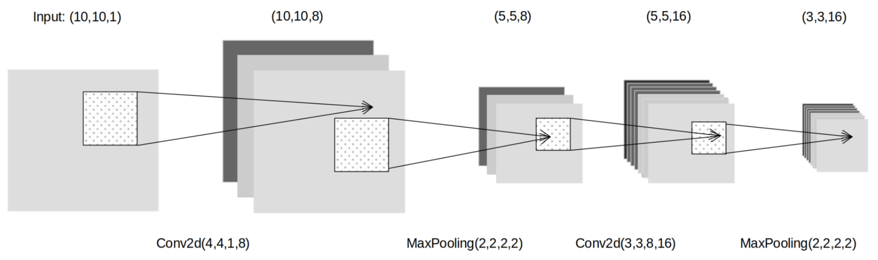
Beyond the causal structure that connects variables, there are two types of background knowledge that we can inject into the rectangle model. First, consider knowledge that manifests as *logical constraints.* Given the row and column of the upper-left corner, we may be able to infer bounds on the heights and widths of the rectangle. For example, if the upper-left corner of a rectangle is on the third row, then its height is at most 8. We can thus fix the corresponding CPT parameters in the model: $P(\mathsf{height}=9|\mathsf{row}=3) = P(\mathsf{height}=10|\mathsf{row}=3) = 0$. Similarly, we can fix some of the CPT parameters for both $P(\mathsf{width}|\mathsf{col})$ and $P(\mathsf{shape}|\mathsf{width}, \mathsf{height})$ to zero. Next, consider knowledge that manifests as *functional dependencies,* where the value of a node is uniquely determined by the values of its direct causes. If we consider the CPT parameter $P(\mathsf{row}_i|\mathsf{row}, \mathsf{height})$, we find that the value of the binary variable row_i is uniquely determined by the row of the upper-left corner and the height of the rectangle. In particular, $\mathsf{row}_i = \mathsf{on}$ iff $\mathsf{row} \leq \mathsf{row}_i < \mathsf{row} + \mathsf{height}$ and similarly $\mathsf{col}_j = \mathsf{on}$ iff $\mathsf{col} \leq \mathsf{col}_j < \mathsf{col} + \mathsf{width}$ so each variable col_j is also a function of col and width.

Suppose now that we wish to build a classifier that predicts the rectangle shape. We will generate an AC from the Bayesian network model, where variables pixel_{ij} are the AC inputs and variable shape is its output as shown in [8]. The AC parameters will correspond to Bayesian network parameters: some parameters will be fixed due to background knowledge, others will be learned from labeled data. Background knowledge reduces the number of learned parameters and can be critical for successfully compiling and efficiently training ACs [8].

Digits Model The generative model for seven-segment digits is a composition of the rectangle model; see Figure 3 [8]. If we treat each segment as a rectangle, a seven-segment digit is a combination of seven rectangle models (called segment modules). If we know the row & column of the digit's upper-left corner in addition to its height & width, we can determine the location of each segment. In our model, the height of a digit is always seven and its width is always four. Hence, the row & column of the upper-left corner completely determine the bounding box of the digit in the image, in addition to the location and size of each potential segment. Moreover, once we know the digit's identity we also know which of the seven segments are active. Hence, each variable $\mathsf{active_k}$, $k \in \{1, \ldots, 7\}$, is functionally determined by the variable digit. For each segment module k, we build a sub-module with the same architecture as the rectangle model, except that the values of segment pixels also depend on whether the segment is active; that is, a segment variable $\mathsf{pixel^k_{ij}}$ is turned on only if segment k is active. Finally, each variable $\mathsf{pixel_{ij}}$ for the image is a disjunction of the pixel values for each segment module. That is, $\mathsf{pixel_{ij}} = \mathsf{on}$ iff $\mathsf{pixel^k_{ij}} = \mathsf{on}$ for some segment k. Again, when generating noisy images, the relation between $\mathsf{pixel_{ij}}$ and its direct causes $\mathsf{pixel^k_{ij}}$ is probabilistic and the corresponding conditional distributions are learned from data. The other distributions learned from data are the priors on row, col and digit. Like the rectangle model, the digit model encodes background knowledge in the form of logical constraints and functional dependencies. The compiled AC has variables $\mathsf{pixel_{ij}}$ as its input and variable digit as its output.

5 Convolutional Neural Network (CNN) Classifiers

We use the CNN architecture below for the rectangle and digits classifiers, but with different outputs: 2 for rectangle shape (wide vs. tall), 10 each for rectangle height/width/row/column (values), and 10 for digits (one for each digit). We use "same" convolutions and ReLU activation functions.



The input layer is a 10×10 black-and-white image, then (1) a convolutional layer with eight 4×4 filters with stride 1, followed by batch normalization, (2) a max-pooling layer with 2×2 filters with stride 2, followed by dropout (rate 20%), (3) a convolutional layer with sixteen 3×3 filters with stride 1, followed by batch normalization, (4) a max-pooling layer with 2×2 filters with stride 2, followed by dropout (rate 20% for rectangles, 50% for digits), (5) a fully-connected layer.[3]

6 Classifiers, Data Generation & Training

We conduct experiments using three types of classifiers that we describe next.

[3] We tuned the dropout rate, the number/size of filters and the learning rate. But we don't know of a way to guarantee that this is the best performing neural network.

Table 1. Noise levels defined by rectangle noise, pixel noise, and removal noise.

noise level	null	ignorable	easy	medium	moderate	hard	superhard
rectangle noise	0	2	2	5	7	7	10
pixel noise	0	2	5	5	5	7	10
removal noise	0	0	1	1	1	2	2

• BN **classifier:** An AC compiled from a Bayesian network that does not include logical constraints or functional dependencies (i.e., no known parameters). We do exploit the *existence* of functional dependencies when compiling the network, but without incorporating the corresponding parameters into the AC.[4]

• BN+BK **classifier:** An AC compiled from a Bayesian network that encodes logical constraints and functional dependencies (i.e., some of the network parameters are known). This AC has a smaller number of trainable parameters compared to the AC described above.

• CNN **classifier:** A convolutional neural network as described in Section 5.

The ACs were represented using tensor graphs as described in [8]. The sum of tensor sizes in the graph is reported next as the AC size. The ACs for predicting rectangle shape had the following sizes: $3,540,319$ (BN) and $3,522,136$ (BN+BK). The ACs for predicting digits had the following sizes: $62,185,299$ (BN) and $2,256,646$ (BN+BK).[5] For the rectangle classifiers, the counts of trainable parameters are: $5,220$ (BN), 136 (BN+BK) and $1,642$ (CNN). For the digits classifiers: $48,141$ (BN), 275 (BN+BK) and $2,802$ (CNN). One reason why BN+BK classifiers had such a small number of parameters, beyond integrating background knowledge, is that we *tied* the CPTs of pixels so they share trainable parameters (more on this later).

In what follows, a *clean* image is one generated from the model without adding noise, while a *noisy* image is a clean image to which noise was added using one of the methods described below.

Generating Rectangle Data A rectangle in an image is defined by the row and column of its upper-left corner, in addition to its width and height. Hence, we can generate all clean rectangle images by simply considering all valid combinations

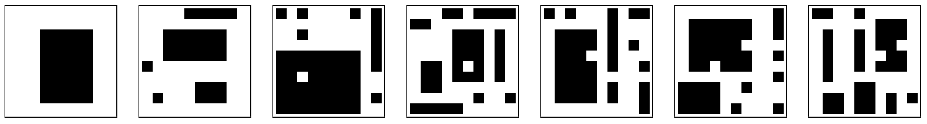
[4] The AC compilation algorithm can computationally exploit that a node X is a function of its parents $\mathbf{U}$, $X = f(\mathbf{U})$, even when the function f is not known; see [8] for details. Exploiting functional dependencies computationally is necessary to successfully compile the digits model due to its very high treewidth. However, since our training algorithm does not guarantee that a functional dependency will be learned, the resulting AC can be viewed as an approximate compilation.

[5] We employed value propagation and pruning techniques when compiling ACs with background knowledge. This is quite effective on the digits model as we can infer that some pixels will never be turned on (the pruning lessens as the image size gets larger). This explains the significant drop in the digits AC size when adding background knowledge and why the digits AC is smaller than the rectangle AC on 10×10 images (this changes for larger image sizes).

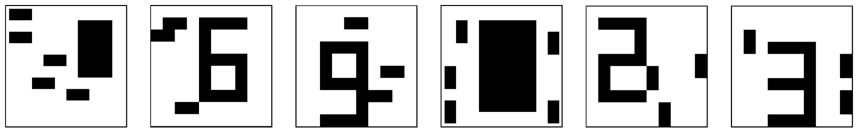
of row, column, width and height (we omit squares). We use a black color for pixels inside the rectangle and a white color for pixels outside the rectangle.

A noisy rectangle image is obtained by adding the following types of noise, in order. First, there are three parameters x, y and z, in addition to the rectangle width W, height H and area $A = W \cdot H$. *Removal noise* (x): we choose a random number k from $[0, x]$ and then flip k random black pixels to white. *Rectangle noise* (y): We add at most y smaller black rectangles to the white background (stop when no white pixels are left). We first select a random position for the upper-left corner. In the available white space, we select a random width and height such that the resulting area is at most $\frac{A-k}{2}$. We further guarantee there is at least one pixel separating each rectangle. *Pixel noise* (z): We flip z random white pixels to black. If no further valid perturbations become possible, we stop early. We assume that x is at most $\min(W, H) - 1$ and z is at most $\min(A - k - 1, \frac{R}{2})$, where R is the count of white pixels left. Table 1 depicts seven levels of noise which result from setting parameters x, y and z. We will use these noise categories later in Section 7.

While the noise generation model may appear complex, it is meant to avoid excessive distortion of the original rectangle and to keep it as the largest rectangle in the noisy image. Otherwise, it would be difficult even for a human to recognize these rectangles. Some example noisy rectangles are depicted below (with increasing noise from left-to-right).



One special type of noise we use is *paired noise:* an adjacent pair of black pixels that we add to the white background of a rectangle image. Paired noise corresponds to a wide or tall rectangle with an area of exactly 2, like in the following examples.



We will use paired noise when assessing classifier robustness in Section 8.

Generating Digits Data Generation of digit images is similar to the generation of rectangles. We first create a clean image of a digit in a random position. We add noisy images by selecting some number $k \in \{0, 2, \ldots, 18\}$ of random white pixels to flip to black.

Training All classifiers were trained using TensorFlow using cross-entropy loss and the Adam optimizer. For each training set, we set aside 20% of the instances for validation (we stop gradient descent if the validation loss does not improve

Table 2. Accuracy of detecting rectangle properties while varying training data size.

	BN Classifier			BN+BK Classifier			CNN Classifier		
Data	Shape	Col	Height	Shape	Col	Height	Shape	Col	Height
25	50.64	29.53	20.35	96.71	87.23	81.88	72.02	32.26	20.72
50	53.17	34.77	19.95	93.12	91.88	86.79	81.73	49.74	24.71
100	56.32	49.67	24.08	94.44	97.37	93.04	83.73	75.21	31.27
250	67.54	63.51	26.49	98.97	98.42	97.26	86.13	84.74	39.09
500	77.92	73.02	31.45	99.82	98.61	97.38	90.05	91.67	61.54
1000	81.20	91.93	69.20	99.98	98.86	97.56	97.12	96.56	84.54
2000	83.40	98.11	91.63	99.99	98.90	97.66	98.37	98.77	92.52
4000	88.99	98.98	98.61	100.00	98.95	97.85	99.18	99.39	96.35
8000	95.44	99.79	99.15	99.99	99.18	98.01	99.64	99.77	97.51

Table 3. Accuracy of classifying digits while varying the size of training data.

Training Data	25	50	100	250	500	1000	2000	4000	8000
BN	12.68	11.66	17.42	23.57	32.19	42.49	56.35	84.33	91.49
BN+BK	72.07	96.39	98.52	98.56	98.64	98.83	99.10	99.07	99.14
CNN	30.34	29.22	43.45	78.79	95.13	97.29	97.97	98.48	98.48

enough). A separate testing set is used to evaluate classifier accuracy. We report the best test accuracy obtained from five different runs with random seeds.

For AC classifiers, we used the PyTAC system which compiles Bayesian networks into ACs, using TensorFlow for training [8]. We used early stopping, a learning rate with polynomial decay over 100 steps and a batch size of 32 (we start at a rate of 0.05 and end at 0.005, using a polynomial power of 3). For neural network classifiers, we used a learning rate with exponential decay, starting at 0.01 with a decay of 0.95 over the first 5,000 batches, using a batch size of 64. Across 5,000 epochs, we saved the CNN with the best validation loss.

7 Sensitivity of Classifiers to Data Size and Quality

We next assess the impact of size/quality of training data on classifier performance. Our first experiment examines how many labeled images we need to get certain prediction accuracies. Our second experiment assesses the sensitivity of classifiers to the ratio of clean to noisy images in training data.

Dataset Size We first consider classifier performance as a function of training data size. For the rectangle model, both training and testing images are 20% clean with a medium noise level for all noisy images. We vary the training dataset size from 25 to 8,000 images and report the accuracy of the learned classifiers on a randomly generated testing dataset of size 10,000. We increment the size of training data in the same way for the digits model. We use 1% clean images in both training and testing data and inject 10 noisy pixels when producing noisy images (there is a limited number of distinct clean images for digits on a 10×10 grid). We report classifier accuracy on a testing dataset of size 28,000.

Table 4. Standard deviations of classification accuracies for detecting rectangle shape (top) and recognizing digits (bottom), while varying the size of training data.

Training Data	25	50	100	250	500	1000	2000	4000	8000
BN	0.92	1.26	1.73	3.10	3.78	2.22	2.46	1.78	2.19
BN+BK	10.93	16.93	5.75	1.19	0.89	0.23	0.32	0.03	0.03
CNN	5.73	6.01	2.65	2.34	0.29	0.98	0.41	0.19	0.09

Training Data	25	50	100	250	500	1000	2000	4000	8000
BN	1.45	1.17	2.46	3.48	2.17	0.78	2.68	3.50	1.72
BN+BK	24.69	1.90	1.25	0.40	0.12	0.21	0.20	0.10	0.08
CNN	5.92	2.53	6.46	2.67	0.72	0.94	0.39	0.14	0.17

The accuracies of all classifiers are shown in Tables 2 and 3 (we omit results for classifying Row and Width for space limitations). The accuracy increases when the size of training data increases for all classifiers. BN+BK gives very accurate predictions compared to the other two classifiers: The presence of background knowledge improves the accuracy significantly especially for small training data. For instance, for a dataset of size 250, BN+BK significantly outperforms BN and CNN. The latter outperforms BN in most cases and reaches a comparable performance to BN+BK when the training data is large enough.

In Table 4, we report some of the standard deviations for accuracies obtained in our experiments. When the training dataset size is ≥ 1000, the standard deviations of BN+BK and CNN are decreasing in general and smaller than the standard deviations of BN. Both BN+BK and CNN have larger standard deviations than BN when the number of examples is small (≤ 100), despite their potential to attain higher accuracies with a small dataset.

Data Quality We now consider how the percentage of noisy images in training data affects the performance of rectangle classifiers. We fix the size of training data to 2,000 and the noise level to medium but vary the percentage of noisy images. We test on 10,000 images that are 20% clean. Table 5 depicts the results. Classification accuracy generally decreases as we decrease the number of noisy images in training data. This is evident for the BN and CNN classifiers whose performance rely non-trivially on having a good number of noisy images during training. However, the BN+BK classifier exhibits much less sensitivity (almost insensitive) and continues to predict very accurately even as we train it on a significantly reduced number of noisy images (10%).

8 Robustness of Classifiers

We next assess the robustness of classifiers by varying the amount and type of noise in testing data. Our first experiment investigates classifier performance on testing images with noise levels that are different from those in the training data. Our second experiment keeps the noise level in testing data similar to that in training data but changes its nature.

Table 5. Classifier accuracy for different percentages of noisy images in training data.

	BN Classifier			BN+BK Classifier			CNN Classifier		
Noisy%	Shape	Col	Height	Shape	Col	Height	Shape	Col	Height
90	85.37	95.07	89.03	99.94	98.89	97.93	98.71	99.04	91.59
80	85.57	97.84	91.02	99.97	98.95	97.67	98.29	98.74	89.72
70	84.15	93.84	88.76	99.96	98.83	97.76	98.10	98.37	90.31
60	84.42	95.21	90.58	99.89	99.06	97.55	97.61	97.52	89.37
50	84.37	88.99	88.88	99.99	98.87	97.76	97.75	96.80	89.04
40	84.11	85.61	85.97	99.91	98.65	97.55	96.58	96.25	87.68
30	80.97	79.00	84.50	99.95	98.81	97.62	95.53	95.43	87.57
20	81.69	84.68	79.48	99.79	99.02	97.42	94.11	91.54	81.26
10	78.41	77.45	74.88	99.95	98.81	97.38	93.04	88.69	74.88

Table 6. Classifier accuracy while varying noise level in testing data.

	BN Classifier			BN+BK Classifier			CNN Classifier		
Noise	Shape	Col	Height	Shape	Col	Height	Shape	Col	Height
Null	93.16	100.00	99.08	100.00	100.00	100.00	100.00	100.00	99.16
Ignorable	88.25	98.74	96.61	99.98	99.91	99.84	99.02	99.48	96.21
Easy	87.30	97.99	95.30	99.98	98.97	98.27	99.12	99.35	95.67
Medium	85.28	96.15	92.51	99.98	98.97	97.65	98.37	98.53	93.39
Moderate	83.61	94.81	89.83	99.98	98.57	97.72	98.07	97.95	91.48
Hard	83.40	93.87	90.09	99.89	98.42	97.15	97.92	98.03	90.82
Superhard	82.22	91.80	88.24	99.89	98.30	96.56	97.17	97.30	87.93

8.1 Noise Level in Testing Data

For the rectangle model, we first train classifiers on 2,000 labeled images that are 20% clean and use a medium noise level for images. We then test the trained classifiers on 2,000 labeled images that are 20% clean but with varying noise levels (see Table 1 for the definitions of noise levels). Table 6 depicts the obtained prediction accuracies. A similar experiment is conducted for the digits model. We also train classifiers using 2,000 labeled images that are 1% clean. All noisy images in the training dataset contain 8 noisy pixels. We then generate 28,000 testing images that are also 1% clean while increasing the amount of noise in testing images. Table 7 depicts the obtained accuracies.

According to the tables, the accuracy for BN+BK and CNN is highest (close to 100%) when little noise is present in the testing dataset. Among the three classifiers, BN has the fastest drop in accuracy when more noise is injected in the testing data than in the training data. BN+BK is the least affected as it has a slight drop in accuracy even when the testing data is much noisier than the training data. For the digits experiment, we also observe that BN performs better when the level of noise seen in testing is more comparable to the level of noise seen in training. In general, the experiments show that background knowledge can significantly improve the robustness of a classifier.

Table 7. Accuracy of classifying digits while varying noise level in testing data.

# noisy pixels	0	2	4	6	8	10	12	14	16	18
BN	48.21	57.77	65.44	68.62	65.61	57.31	48.23	41.52	35.24	29.85
BN+BK	100.00	99.99	99.86	99.70	99.38	99.00	98.40	97.69	97.03	96.12
CNN	100.00	99.95	99.79	99.33	98.74	97.84	96.46	93.97	91.24	86.98

Table 8. Accuracy of classifying rectangles (left) and digits (right) for 10×10 images when the training–testing data have different types of paired noise.

Classifier	horizontal-vertical			vertical-horizontal		
	Shape	Col	Height	Shape	Col	Height
BN	89.95	84.67	86.68	88.16	92.14	86.38
BN+BK	99.81	100.00	99.06	99.82	99.80	100.00
CNN	66.60	83.77	67.83	83.52	78.04	46.83

horizontal-vertical	vertical-horizontal
74.03	70.55
97.25	98.97
90.16	81.16

8.2 Discrepancy Between Training and Testing Data

In our second experiment, we use different types of paired noise in training and testing data; see Section 6 for the definition of paired noise. There are two combinations: train with horizontal paired noise and test with vertical paired noise, or vice versa. We train and test all classifiers under these combinations, using 2,000 labeled images that are 20% clean for rectangle classifiers and 2,000 labeled images that are 1% clean for digits classifiers. We test the rectangle classifiers on 10,000 images that are 20% clean and the digits classifiers on 28,000 images that are 1% clean. The results are shown in Table 8. BN+BK classifiers, which encode background knowledge, have the best performance by a very large margin. The stark difference in robustness revealed by this simple experiment strongly highlights the promise of supervised learning with background knowledge.

Even though we reported only on 10×10 images, we could compile ACs with background knowledge for 30×30 rectangle images (900 AC inputs) and for 22×22 digits images (484 AC inputs). But these ACs were too large to be trained consistently and effectively by a basic application of gradient descent on a CPU (the first AC had 21,899 tensor operations and size 680.6M, and the second AC had 77,702 tensor operations and size 2,552.5M). Moreover, ACs for these larger images would not compile without background knowledge. For the experiment in this section though, we managed to train ACs with background knowledge on 16×16 images (AC size is 58.8M) and on 20×20 images (AC size is 224.2M). For comparison, we also ran experiments on CNN classifiers with the same architecture discussed earlier with the corresponding image sizes. The results, reported in Table 9, are similar to what we reported for 10×10 images in Table 8: AC classifiers maintained an accuracy $\geq 99\%$ but CNN classifiers basically failed with the accuracy dropping to as low as 42% in some cases. For reference, the number of trainable parameters for 20×20 images were 466 (AC) and 2,154 (CNN) for predicting rectangle shape. All of our experiments were run on a 2.60GHz Intel Xeon E5-2670 CPU with 256 GB of memory.

Table 9. Accuracy of classifying rectangles when the training–testing data have different types of paired noise. This experiment uses larger images than the one in Table 8.

Classifier		horizontal-vertical			vertical-horizontal		
		Shape	Col	Height	Shape	Col	Height
16x16	BN+BK	99.80	99.64	99.19	99.62	99.68	99.74
	CNN	71.27	87.72	64.98	67.85	85.60	46.58
20x20	BN+BK	99.74	99.59	99.60	99.70	99.60	99.61
	CNN	75.06	87.21	52.46	65.00	88.05	42.04

9 Challenges and Opportunities

The results we reported show the promise of model-based supervised learning: it can be significantly less data hungry and can lead to much more robust classifiers. This is not surprising given that model-based classifiers "understand" the objects they are classifying compared to model-free classifiers which settle once they have captured the patterns in data well enough (a process that may not lead to truly learning a task as we have seen particularly in the last experiment). What is perhaps surprising is the extent of the gap sometimes revealed in the experiments we conducted. This all, however, begs the question: Why is model-based supervised learning not as commonly used in practice and why is model-free supervised learning so dominant despite its limitations?

As mentioned in the introduction, model-based supervised learning faces a number of barriers. Some are profound and require further theoretical developments and some are more mundane, resulting from a lack of enough efforts in certain directions. Two of the profound barriers are: the acquisition of accurate models and the compilation of models into circuits. The first barrier has been discussed in [28] and an initial treatment has already been proposed in [5, 4]. In a nutshell, these works show that incorrect models can lead to ACs that are not expressive enough to capture the true classifier. Moreover, the proposed treatment is based on using Testing Arithmetic Circuits (TACs), which represent piecewise distributions or mixtures of distributions while maintaining the guarantees offered by the underlying causal models. This is a promising direction as it does not insist on obtaining a fully accurate model, which can be practically impossible in some situations. One would need to advance this or similar directions further if model-based supervised learning is to become more practical.

The second profound barrier relates to the compilation of ACs from models. This is an inference task at heart and as such requires fundamental algorithmic developments. Much progress has been made on this front over the years, but more progress is needed. The two case studies we discussed provide a concrete context for relaying this particular point. Despite the simplicity of the tasks, and the relatively small size of images, the underlying models proved quite challenging to compile. In fact, compiling and efficiently training these ACs was only possible due to the recent results reported in [8]. In this work, functional dependencies were exploited to make the compilation feasible, while preserving two properties that are critical in a learning context. First, the resulting ACs were easily representable as tensor graphs since the proposed compilation algorithm was based on the use of dense factors. This facilitated the application of gradient descent in a style similar to what is done when training neural networks, leading to orders of magnitude reduction in training time. Second, the compilation algorithm exploited functional dependencies computationally without needing to know their identity, therefore allowing one to learn such dependencies from data. This is a stark difference with earlier compilation approaches which required knowledge of the specific functional dependencies. This work is a first step and more work is needed along these directions: the computational exploitation of background knowledge and tensor-based technologies.

Another barrier relates to the application of gradient descent to ACs that are compiled from models, which tend to have structures and behaviors that are different from those of neural networks. As a result, the know-how and common wisdom gathered through much research on training neural networks is not immediately applicable to training such ACs. For example, for the considered tasks, ACs are significantly larger in size, have many more layers (i.e., deeper), and have a much larger compute-to-parameter ratio, all calling for a dedicated study of gradient descent algorithms in this particular context. Even primitive tasks, like how to seed gradient descent algorithms, have barely been touched while much research has been done on this front when training neural networks. It is worth mentioning two anecdotes on this front. First, given the lack of principled methods for seeding AC parameters, we ended up trying five seeds for each run: one uniform seed and four random seeds. We then did a two-step lookahead on each seed and choose the best to use as a seed for gradient descent, leading to better results than initially reported in [8]. Despite this improvement, we still see more outlier runs in comparison to what one sees when training neural networks. Second, when we experimented with tied CPTs for pixel nodes in BN+BK classifiers, our initial goal was to reduce the strain on gradient descent by having it learn a smaller number of parameters. Surprisingly, for the investigated tasks, this not only enhanced the speed of gradient descent but yielded better accuracies as reported earlier. We also ran comparisons between BN and BN+BK, both with and without tying parameters, which we omit for space. Briefly, incorporating background knowledge improves accuracy, whether we tie parameters or not. Generally, tying parameters improves computational performance, as well as accuracy, whether we incorporate background knowledge or not. However, improvements in accuracy were more consistent when incorporating background knowledge rather than not. These observations suggest that gaps in performance, whether we incorporate background knowledge or not, or whether we tie parameters or not, could be partially due to gradient descent struggling with the larger number of parameters it has to learn in each case. Again, this all calls for further research on the application of gradient descent to ACs that are compiled from models.

To summarize and close: Advancing model-based supervised learning not only requires further theoretical developments, but it also requires reducing the know-how gap with the model-free community as far as gradient descent and the exploitation of tensor-based technologies. This requires a broad and sustained enough effort by the model-based community which we hope it will heed.

Bibliography

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

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

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

[4] Choi, A., Darwiche, A.: On the relative expressiveness of Bayesian and neural networks. In: PGM. pp. 157–168 (2018)

[5] Choi, A., Wang, R., Darwiche, A.: On the relative expressiveness of Bayesian and neural networks. International Journal of Approximate Reasoning **113**, 303–323 (2019)

[6] Darwiche, A.: A differential approach to inference in Bayesian networks. Journal of the ACM (JACM) **50**(3), 280–305 (2003)

[7] Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press (2009)

[8] Darwiche, A.: An advance on variable elimination with applications to tensor-based computation. In: ECAI (2020), `https://arxiv.org/abs/2002.09320`

[9] Elidan, G., Friedman, N.: Learning hidden variable networks: The information bottleneck approach. J. Mach. Learn. Res. **6**, 81–127 (2005)

[10] van Engelen, R.A.: Approximating Bayesian belief networks by arc removal. PAMI **19**(8), 916–920 (1997)

[11] Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29**(2-3), 131–163 (1997)

[12] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)

[13] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation **18**(7), 1527–1554 (2006)

[14] Jaeger, M.: Probabilistic decision graphs — combining verification and AI techniques for probabilistic inference. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **12**, 19–42 (2004)

[15] Kjærulff, U.: Reduction of computational complexity in Bayesian networks through removal of weak dependences. In: UAI. pp. 374–382 (1994)

[16] Lowd, D., Domingos, P.M.: Learning arithmetic circuits. In: Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI). pp. 383–392 (2008)

[17] Mateescu, R., Dechter, R., Marinescu, R.: AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. J. Artif. Intell. Res. (JAIR) **33**, 465–519 (2008)
[18] Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Mauro, N.D., Poupart, P., Kersting, K.: SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks (2019)
[19] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. MK (1988)
[20] Pearl, J.: Causality. Cambridge University Press (2000)
[21] Pearl, J., Mackenzie, D.: The Book of Why: The New Science of Cause and Effect. Basic Books (2018)
[22] Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., den Broeck, G.V., Kersting, K., Ghahramani, Z.: Einsum networks: Fast and scalable learning of tractable probabilistic circuits. CoRR **abs/2004.06231** (2020)
[23] Ranzato, M., Poultney, C.S., Chopra, S., LeCun, Y.: Efficient learning of sparse representations with an energy-based model. In: NIPS. pp. 1137–1144 (2006)
[24] Russell, S.J., Binder, J., Koller, D., Kanazawa, K.: Local learning in probabilistic networks with hidden variables. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, (IJCAI). pp. 1146–1152 (1995)
[25] Sanner, S., McAllester, D.A.: Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI). pp. 1384–1390 (2005)
[26] Schwalb, E.: Compiling bayesian networks into neural networks. In: Proceedings of the Tenth International Conference on Machine Learning (ICML). pp. 291–297 (1993)
[27] Shen, Y., Choi, A., Darwiche, A.: Tractable operations for arithmetic circuits of probabilistic models. In: Advances in Neural Information Processing Systems 29 (NIPS) (2016)
[28] Shen, Y., Huang, H., Choi, A., Darwiche, A.: Conditional independence in Testing Bayesian Networks. In: ICML. pp. 5701–5709 (2019)
[29] Suermondt, H.J.: Explanation in Bayesian Belief Networks. Ph.D. thesis, Stanford (1992)
[30] Thiesson, B., Meek, C., Heckerman, D.: Accelerating EM for large databases. Mach. Learn. **45**(3), 279–299 (2001)