Deep Logic Networks: Inserting and Extracting Knowledge From Deep Belief Networks

Son N. Tran and Artur S. d'Avila Garcez

***Abstract*— Developments in deep learning have seen the use of layerwise unsupervised learning combined with supervised learning for fine-tuning. With this layerwise approach, a deep network can be seen as a more modular system that lends itself well to learning representations. In this paper, we investigate whether such modularity can be useful to the insertion of background knowledge into deep networks, whether it can improve learning performance when it is available, and to the extraction of knowledge from trained deep networks, and whether it can offer a better understanding of the representations learned by such networks. To this end, we use a simple symbolic language—a set of logical rules that we call *confidence rules*—and show that it is suitable for the representation of quantitative reasoning in deep networks. We show by knowledge extraction that confidence rules can offer a low-cost representation for layerwise networks (or restricted Boltzmann machines). We also show that layerwise extraction can produce an improvement in the accuracy of deep belief networks. Furthermore, the proposed symbolic characterization of deep networks provides a novel method for the insertion of prior knowledge and training of deep networks. With the use of this method, a deep neural–symbolic system is proposed and evaluated, with the experimental results indicating that modularity through the use of confidence rules and knowledge insertion can be beneficial to network performance.**

***Index Terms*— Deep belief networks (DBNs), deep learning, knowledge extraction, knowledge representation and reasoning, neural–symbolic integration.**

I. Introduction

KNOWLEDGE representation and reasoning are two important topics in artificial intelligence (AI). Since the early years of AI, researchers have sought to represent knowledge in symbolic form to facilitate automated reasoning and help explain a machine's conclusions much like humans do. The benefits of having a symbolic representation are that knowledge can be verified formally, consolidated into knowledge bases that can offer insight into the nature of the problem domain, and used to provide explanations for the answers produced by the system, e.g., in the form of proofs.

Although symbolic representations have been deployed successfully in many intelligent and multiagent systems, flexible representations and forms of reasoning are needed that can account for the inherent uncertainty in the problems tackled by AI [1], [2], such as, for example, handling noise in video and audio data. In such domains, statistical models such as deep networks seem capable of approximating a desired input–output function well purely from data for application in a range of classification tasks. More specifically, a deep belief network (DBN) [3] can be constructed by stacking several generative restricted Boltzmann machines (RBMs) [4]. Nevertheless, scientists are still concerned about offering a better understanding of how such complex systems work [5].

Within the area of neural–symbolic computing [6], [7], efforts have been placed on how knowledge can be inserted and extracted from neural network models [8], [9]. Background knowledge insertion, whenever it is available, is expected to offer an alternative to network pretraining. Knowledge extraction after training is expected to provide explanations [10]–[14], as already mentioned. A major challenge that has not been addressed fully yet is how to build an efficient neural–symbolic system capable of capturing the semantic meaning of a domain within a simple representation while supporting flexible inference for the given large-scale noisy data, particularly from image, video, and audio data.

Results in deep learning [3], [15] have shown that the combination of unsupervised (layerwise modular) network training with supervised training (also known as fine-tuning, using backpropagation) over multiple network layers can provide an effective way of learning and representing complex patterns. Bottou [16] argues that modularity should be the key to building a reasoning system that combines rich inference and simple manipulations from linked models.

In this paper, we study the effects of modularity on a hierarchical network from a neural–symbolic perspective. We propose and evaluate methods and algorithms for inserting hierarchical knowledge into deep networks and for knowledge extraction from deep networks following network training [9]. A simple symbolic language for hierarchical knowledge representation, named *confidence rules* [17]–[19], is deployed and evaluated. Confidence rules are shown empirically to be suitable for combining symbolic knowledge and quantitative reasoning, including probabilistic inference in deep networks. Each confidence rule is associated with a real number called a *confidence value* c such that a rule of the form $c : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3$ denotes that h holds with confidence c if x_1 holds, x_2 does not hold, and x_3 holds. In order to perform inference under uncertainty dealing with continuous-valued data as well as missing values, an extension of the

Manuscript received March 19, 2016; revised July 1, 2016; accepted August 4, 2016. *(Corresponding author: Artur S. d'Avila Garcez.)*

The authors are with the Department of Computer Science, City University London, London, EC1V 0HB, U.K. (e-mail: son.tran.1@city.ac.uk; aag@soi.city.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2016.2603784

traditional *modus ponens* logical inference rule is introduced. A novel and efficient knowledge extraction algorithm is then shown capable of deriving useful confidence rules from DBNs trained directly from data. Finally, we study how to insert prior knowledge into deep networks. In [20] and [21], prior knowledge such as local information of image batches was used to improve learning of deep networks. In this paper, we study the effects of inserting hierarchical knowledge into deep networks. We introduce a deep neural–symbolic system capable of encoding symbolic knowledge into DBNs and leveraging the above inference algorithm to guide and improve learning of the model.

This paper is organized as follows. The next section reviews the related literature in DBNs and neural–symbolic integration. Section III defines confidence rules and shows their relationship with deep networks. Section IV proposes and evaluates the general algorithm for extracting symbolic knowledge from RBMs and extends it to deal with DBNs. Section V shows how prior knowledge in the form of hierarchical confidence rules can be encoded effectively into deep networks and how this can improve learning performance in comparison with DBNs. Section VI concludes this paper and discusses directions for future work.

II. Preliminaries

A. Notation

Matrices and vectors are denoted using boldface capital letters $\mathbf{X}$ and boldface letters $\mathbf{x}$, respectively. A subscript is used to denote an element of a matrix or vector, for example, x_{ij} and x_i. A vector $\mathbf{x}_j$ denotes the column j of matrix $\mathbf{X}$. While numerical variables are denoted as x, a logical proposition is denoted as $\mathbf{x}$, and it assumes two possible values: $true$ or $false$, which are equivalent to a binary variable x with values 1 and 0, respectively.

B. Deep Networks

In unsupervised learning, deep networks [3], [15], [22] have been established as a powerful architecture for the learning of features at different levels of abstraction. For example, a DBN [3] can be constructed by stacking several generative RBMs [4] one on top of another in order to learn better representations of unlabeled data. Shallow generative networks with a single hidden layer such as RBMs have an advantage that inference, i.e., the conditional probability of a unit's state, given the states of the other units, can be computed exactly. However, they are limited in their learning to a single level of representation [15]. Deep architectures [3], [20], [23]–[25], on the other hand, are connectionist systems that consist of many hidden layers and therefore are capable of learning representations at multiple levels of abstraction. Even though these deep models are complex in terms of learning, e.g., their cost function is nonconvex, research has shown that they can scale better than their shallow counterparts [15].

Formally, one can represent a DBN as a hierarchy of network layers $X, H^{(1)}, \ldots, H^{(L)}$, where X is a visible layer and $H^{(l)}$ $(l = 1 : L)$ are hidden layers. Let us consider a network with a single hidden layer ($L = 1$; i.e., an RBM [4], [26]). This two-layer connectionist model is characterized by an energy function

$$E(\mathbf{x}, \mathbf{h}^{(1)}) = -\sum_{ij} x_i w_{ij} h_j^{(1)} - \sum_i a_i x_i - \sum_j b_j h_j^{(1)} \quad (1)$$

where $w_{ij} \in \mathbf{W}$ is the weight between unit i in layer X and unit j in layer $H^{(1)}$ and a_i and b_j are the biases of the units in X and $H^{(1)}$, respectively. For learning, one can train the model by maximizing the log-likelihood

$$\log P(\mathbf{x}) = \log \frac{\sum_{\mathbf{h}^{(1)}} e^{-\mathbf{E}(\mathbf{x}, \mathbf{h}^{(1)})}}{Z} \quad (2)$$

where Z is the partition function $Z = \sum_{\mathbf{x}, \mathbf{h}^{(1)}} e^{-E(\mathbf{x}, \mathbf{h}^{(1)})}$.

The gradient of this function, call it $\nabla\theta$, can be computed as follows:

$$\nabla\theta = \mathbb{E}\left[\frac{\partial \mathbf{E}(\mathbf{x}, \mathbf{h}^{(1)}; \theta)}{\partial \theta}\right]_{\mathbf{h}^{(1)}|\mathbf{x}} - \mathbb{E}\left[\frac{\partial \mathbf{E}(\mathbf{x}, \mathbf{h}^{(1)}; \theta)}{\partial \theta}\right]_{\mathbf{x}, \mathbf{h}^{(1)}}. \quad (3)$$

Here, we get into a common problem in statistical machine learning: even though the first term can be computed exactly, the second term is intractable as the complexity of computing the joint probability $P(\mathbf{x}, \mathbf{h}^{(1)})$ increases exponentially with the size of $\mathbf{x}$ and $\mathbf{h}^{(1)}$. This is typically a serious problem because most real-world applications such as video analysis will have a large number of attributes $\mathbf{x}$. Now, if we generalize the model into a deep network with multiple hidden layers, the problem of intractability worsens with the dependency between hidden layers to the point that one becomes unable to compute the conditional probability $P(\mathbf{h}^{(l)}|\mathbf{x})$ exactly from the data.

The common way to avoid the above intractability is to learn a deep network layer by layer [3], [24], [27]. To this end, the deep network is used as a stack of several shallow networks, each having a single hidden layer. Training of each layer is performed bottom-up with the focus now turned to how (2) can be solved approximately. This is called layerwise pretraining. After that, the fine-tuning phase trains the network as a whole using the parameters that have been initialized by the pretraining phase. In this paper, we are interested in investigating such a modularity property and the hierarchical reasoning that takes place within deep networks. Therefore, we focus on layerwise training as a key component in learning complex models. We exclude the fine-tuning phase from our models, since it can break the modularity of the models by treating them as traditional multilayer neural networks [15].

C. Neural–Symbolic Systems

Considerable research has been devoted to the integration of symbolic knowledge and connectionist systems [6], [8], [9], [17], [28]–[32]. The first reason for this is that symbolic rules can represent knowledge in a formal language and therefore offer a formal semantics to models and systems. The second is that one may find symbolic knowledge helpful when seeking a better understanding of the connectionist models learned or when seeking to add prior knowledge to such models. Furthermore, symbolic knowledge extracted from

a connectionist model can be employed as a foundation for some other subareas of AI, e.g., knowledge-based transfer learning [33].

In several circumstances, prior knowledge can be provided by domain experts, frequently in the form of symbolic logic rules. Such a use of *domain knowledge* has been shown capable of improving model learning. In [34], a model named knowledge-based artificial neural networks (KBANN) was proposed. KBANN uses multilayer feedforward networks and a method for encoding rules into the networks to enable learning from data and background knowledge. In [35], the connectionist inductive logic programming (CILP) system was introduced. Inspired by KBANN, CILP uses logic programming rules applied to recurrent neural networks to achieve an improvement in performance in comparison with KBANN at learning from data and background knowledge.

Symmetrical systems such as Markov networks and recurrent temporal RBMs (RTRBMs) also have been used for neural–symbolic integration. In [36], a method is presented for encoding background knowledge into a template Markov network [named Markov logic network (MLN)], which is used to create a ground Markov network representing the relationships among all the instances in the data. The idea of representing each formula into a clique of Markov network is similar to that of penalty logic, which has been proposed to integrate symbolic knowledge and Hopfield networks [32]. The difference is that in MLNs, a feature is defined by the number of true groundings of the formulas corresponding to a clique in the template model, while in penalty logic [32], a feature is defined by the multiplication of the variables in the clique. In practice, MLNs work well in a variety of relational domains; however, the models learned are not as comprehensible as one would expect from a symbolic model due to the size of the ground Markov network and exponential nature of such grounding. A more recent development in neural–symbolic integration is the neural–symbolic cognitive agent (NSCA) introduced in [17] in which a model based on RTRBMs is proposed to represent temporal symbolic knowledge and applied to online learning and reasoning. The NSCA model contains algorithms for learning and extraction of temporal logic rules by sampling the RTRBM. It has been applied successfully to driving assessment and training in simulators.

For decades, neural networks have been used successfully as a learning model, from which symbolic rules can be extracted through the use of knowledge extraction algorithms [8], [9], [29]. However, most extraction algorithms exist for discriminative models that do not support modularity. We argue that the modularity found in deep networks may facilitate knowledge extraction, in particular improving the efficiency of extraction from large networks. Most such discriminative extraction approaches treat the class variables as a special type of variable. As a result, the rules that are extracted may be helpful at explaining the relationships between all other variables and the class variables, but not the relationships that might exist among such other variables. For example, suppose a discriminative neural network was trained perfectly to learn the XOR function (denoted by $\oplus$ below) from its truth table. Discriminative knowledge extraction might produce the rule $x_3 \leftrightarrow x_1 \oplus x_2$, with x_3 as the class variable, but might fail to capture equally valid rules $x_1 \leftrightarrow x_2 \oplus x_3$ and $x_2 \leftrightarrow x_1 \oplus x_3$. In what follows, we introduce an efficient method for symbolic knowledge extraction from generative models, specifically RBMs, which can capture such relations. Efficiency is measured by the computational complexity of the rule extraction, as discussed in Section IV, which instead of depending on the combinations of the input vectors (which might be intractable for real-valued inputs or a large number of attributes $\mathbf{x}$) uses the network structure for extraction, depending on the values of the network's weight vectors $\mathbf{W}$ structured hierarchically. Knowledge extraction from DBNs will then be performed layer by layer through the extraction of confidence rules from each RBM [17]–[19].

TABLE I
TRUTH TABLE FOR BICONDITIONAL (IF-AND-ONLY-IF)

x	y	$(x \leftrightarrow y)$
$false$	$false$	$true$
$false$	$true$	$false$
$true$	$false$	$false$
$true$	$true$	$true$

III. CONFIDENCE RULES

Classical logical reasoning has been found difficult to adapt to account for uncertainty in complex domains, with various nonclassical ways of integrating logic and probabilities having been proposed toward this end (for a recent account, see [37]). One way of accounting for uncertainty has been by assigning a real value between 0 and 1 (a probability) to each logical variable so that if, say, x is $false$, given the truth table shown in Table I, one can conclude that the probability of y being $true$ is 0.5.

In what follows, we define confidence rules that will be used for combining symbolic representations and deep networks. Confidence rules support the above form of quantitative inference, useful for reasoning under uncertainty in a way similar to penalty logic [32]. In order to perform inference using confidence rules, dealing with continuous data and missing values, we introduce an algorithm that extends the standard *modus ponens* logical inference rule, as detailed in the following.

A. Confidence Rules

A *confidence rule* is a biconditional (if-and-only-if) formula of the form $c : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \cdots \wedge \mathsf{x}_n$, where h is an atomic proposition and each x_i, $1 \leq i \leq n$, is a literal (an atomic proposition or its negation), labeled by a real-valued number c called a *confidence value* [17]. For example, the formula

$$1.5 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3 \tag{4}$$

associates hypothesis (hidden unit) h with beliefs (visible units) $\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3$ with confidence value 1.5. The *confidence value* as defined in this paper is similar to *confidence credibility* used by NSCAs [17], the *penalty* used by penalty logic [32], and *formula weights* from MLNs [36]. However, a main difference from all of these is that in this paper, confidence values are used for quantitative inference by selecting

INFERENCE RULE: INF1

Given:
$c : \mathsf{h} \leftrightarrow \bigwedge_{\forall t \in T} \mathsf{x}_t \wedge \bigwedge_{\forall k \in K} \neg \mathsf{x}_k$
$\alpha_{t'} : \mathsf{x}_{t'}$ where $t' \in T, \alpha_{t'} \in [min, max]$
$\alpha_{k'} : \mathsf{x}_{k'}$ where $k' \in K, \alpha_{k'} \in [min, max]$
Infer:
$\alpha_h : \mathsf{h}$ with $\alpha_h = c \cdot (\sum_{t'} \alpha_{t'} + \sum_{m \neq \forall t'}^{m \in T} \alpha_m - \sum_{k'} \alpha_{k'} - \sum_{m \neq \forall k'}^{m \in K} \alpha_m)$
where $\alpha_m = \frac{min+max}{2}$

TABLE II
HIERARCHY OF RULES USED AS BACKGROUND KNOWLEDGE FOR THE PROMOTER DNA DATA SET; THE FIRST FOUR RULES APPEAR IN LEVEL L_1 OF THE HIERARCHY, THEN LEVEL L_2, AND SO ON. EACH LEVEL WILL BE MAPPED ONTO A LAYER OF A DBN

L-1{	minus_{35}	$\leftarrow \mathsf{t}_{-36} \wedge \mathsf{t}_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{a}_{-33} \wedge \mathsf{c}_{-32}$
	minus_{10}	$\leftarrow \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{t}_{-7}$
	conformation	$\leftarrow \mathsf{a}_{-45} \wedge \mathsf{a}_{-44}$
	conformation	$\leftarrow \mathsf{a}_{-44} \wedge \mathsf{a}_{-41}\}$
L-2{	contact	$\leftarrow \mathsf{minus}_{35} \wedge \mathsf{minus}_{10}$ }
L-3{	promoter	$\leftarrow$ contact $\wedge$ conformation}

rules for extraction based on their confidence. Specifically, given the confidence values of two formulas, one can quantitatively decide, because of the way that the values will be calculated, which formula should be selected for extraction. In what follows, we describe the inference algorithm for confidence-logic rules.

B. Hierarchical Inference

Many logic programming systems have hierarchical rules in which *intermediate literals* exist. An intermediate literal is a literal that appears in the antecedent (or body) of some rule and in the consequent (or head) of another rule. For example, in the background theory of the DNA promoter data set used in [8] (Table II), literals minus_{35}, minus_{10}, contact, and conformation are intermediate literals. In what follows, we will use the same promoter data set to exemplify our work. We represent the name of each nucleotide with its position in subscript, e.g., t_{-35} denotes that nucleotide t appears at position -35 in the DNA sequence.

Confidence rules can also be organized into hierarchies and inference is performed bottom-up: for each subset of rules in the hierarchy, the confidence value of each hypothesis in the subset is inferred, given the confidence value of each belief that is present. This value is then normalized before being used for inference, now as beliefs, at the next level of the hierarchy using another subset of the rules. Here, normalization is required to maintain the correspondence between the propositions in the rules and the units of the RBMs. The following definition formalizes this idea.

Definition 1: Let $\beta^{(1)}$ be a set of confidence rules relating a set of beliefs $\mathsf{x}_1, \mathsf{x}_2, \ldots$ and a set of hypotheses $\mathsf{h}_1^{(1)}, \mathsf{h}_2^{(1)}, \ldots$. Let $\beta^{(2)}$ be a set of confidence rules relating hypotheses $\mathsf{h}_1^{(1)}, \mathsf{h}_2^{(1)}, \ldots$ and new hypotheses $\mathsf{h}_1^{(2)}, \mathsf{h}_2^{(2)}, \ldots$. Let $\beta^{(3)}$ be a set of confidence rules relating hypotheses $\mathsf{h}_1^{(2)}, \mathsf{h}_2^{(2)}, \ldots$ and new hypotheses $\mathsf{h}_1^{(3)}, \mathsf{h}_2^{(3)}, \ldots$, and so on. We call $\beta^{(1)}, \beta^{(2)}, \beta^{(3)}, \ldots$ a hierarchical weighted knowledge base.

Algorithm 1 QUANT_INFERENCE

1: Initialize the set of beliefs B in level 1 to $\alpha_i : \mathsf{x}_i^{(1)}$, where each belief has a value $\alpha_i^{(1)}$ ($\alpha_i^{(1)}$ can be seen as the input value of visible unit x_i corresponding to proposition x_i);
2: **for** $l = 1$ **to** L **do**
3: Initialize the set of beliefs B_{next} at the next level to zero, i.e., $0 : \mathsf{x}_j^{(l+1)}$
4: **for** each rule r **in** level l **do**
5: Calculate the confidence value α of the hypotheses in r using INF1;
6: Normalize α;
7: Increase the confidence value of the beliefs in B_{next} that are associated with the hypotheses in r by the normalized α;
8: **end for**
9: Set B = B_{next}
10: **end for**

Given an input to any sequence $\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(i)}$ of a hierarchical weighted knowledge base, local inference can be carried out and results propagated to $\beta^{(i+1)}$. This type of inference can be seen as an extension of *modus ponens* to deal with uncertainty through the calculation of confidence values, which allows logical inference to work with real-valued data types through the application of inference rule INF1, as defined above.

In inference rule INF1, T and K are sets of positive and negative literals, respectively. The *confidence value* α_m of any missing beliefs is the average of an upper bound and a lower bound on the normalized *confidence values* in the rule set. For example, given the rule and beliefs:

1) $1.5 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3$;
2) $1 : \mathsf{x}_1$, normalized to $[0, 1]$;
3) x_2 is missing;
4) $1 : \mathsf{x}_3$, normalized to $[0, 1]$

one uses $\alpha_2 = 0.5$ to derive h with confidence value $1.5 \times (2 - 0.5) = 2.25$.

In this process, with normalization bounded by $[min, max]$, if $\neg \mathsf{x}$ has confidence value α, then x must have confidence value $min + max - \alpha$. Algorithm 1 specifies the inference process.

IV. KNOWLEDGE EXTRACTION FROM DEEP NETWORKS

In this section, we focus on the extraction of knowledge in the form of confidence rules from deep networks. In order to build hierarchies of confidence rules, we employ a layerwise approach [3], [15] to extract rules from each layer of a deep network. In what follows, we introduce the algorithm for

knowledge extraction from RBMs and then we extend it to deep networks. For ease of presentation, we omit the biases; however, the following approach can be extended easily to include biases.

A. Knowledge Extraction From RBMs

We now propose an algorithm for extracting rules from a trained RBM. For each hidden unit j, a confidence rule is extracted of the form

$$c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{\forall t \in T} \mathsf{x}_t \wedge \bigwedge_{\forall k \in K} \neg \mathsf{x}_k \tag{5}$$

where x_t and $\neg \mathsf{x}_k$ represent the literals $x_t = 1$ and $x_t = 0$, respectively. The idea behind the extraction algorithm is that it seeks to find the positive and negative literals and confidence values c_j that minimize information loss, according to the following equation:

$$\mathcal{I}_{loss} = \sum_{ij} \| w_{ij} - c_j \mathbb{I}_j(x_i) \|^2 \tag{6}$$

where c_j is the *confidence value* of rule j corresponding to unit j in a hidden layer and

$$\mathbb{I}_j(x_i) = \begin{cases} 1, & \text{if literal } \mathsf{x}_i \text{ appear in rule } j \\ -1, & \text{if literal } \neg \mathsf{x}_i \text{ appear in rule } j \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

For ease of notation, let us use $s_{ij} = \mathbb{I}_j(x_i)$, such that (6) becomes

$$\mathcal{I}_{loss} = \sum_{ij} \| w_{ij} - c_j s_{ij} \|^2. \tag{8}$$

Since (8) is quadratic, the *confidence values* can be found by setting the derivatives to zero as follows:

$$\begin{aligned} \sum_i 2(w_{ij} - c_j s_{ij}) s_{ij} &= 0, \quad \text{for all } j \\ \sum_i w_{ij} s_{ij} - c_j \sum_i s_{ij}^2 &= 0, \quad \text{for all } j. \end{aligned} \tag{9}$$

From (9), we obtain

$$c_j = \frac{\sum_i w_{ij} s_{ij}}{\sum_i s_{ij}^2}. \tag{10}$$

Since the value of s_{ij} is in the set $\{-1, 0, 1\}$, we have

$$\begin{aligned} \| w_{ij} - c_j s_{ij} \|^2 &= \left\| \text{abs}(w_{ij}) - c_j \frac{s_{ij}}{\text{sign}(w_{ij})} \right\|^2 \\ &= \begin{cases} (\text{abs}(w_{ij}) + c_j)^2, & \text{if } s_{ij} \neq \text{sign}(w_{ij}) \\ (\text{abs}(w_{ij}) - c_j)^2, & \text{if } s_{ij} = \text{sign}(w_{ij}) \\ \text{abs}(w_{ij})^2, & \text{if } s_{ij} = 0. \end{cases} \end{aligned} \tag{11}$$

Here, $\text{abs}(w_{ij})$ and $\text{sign}(w_{ij})$ are functions that return, respectively, the absolute value and sign of w_{ij}, respectively. Since $(\text{abs}(w_{ij}) + c_j)^2 > (\text{abs}(w_{ij}) - c_j)^2$ and $(\text{abs}(w_{ij}) + c_j)^2 > \text{abs}(w_{ij})^2$, the information loss will be minimized if $s_{ij} = \text{sign}(w_{ij})$ or $s_{ij} = 0$. In particular, $s_{ij} = 0$ will minimize the loss function if and only if

$$\begin{aligned} \text{abs}(w_{ij})^2 &\leq (\text{abs}(w_{ij}) - c_j)^2 \\ \frac{c_j}{2} &\geq \text{abs}(w_{ij}). \end{aligned} \tag{12}$$

Algorithm 2 RBM_EXTRACT

Require: An RBM with visible layer V and hidden layer H

1: **for** $j = 1$ to the number of hidden units **do**
2: Create rule r_j of the form $c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{w_{tj} > 0} \mathsf{x}_t \wedge \bigwedge_{w_{kj} < 0} \neg \mathsf{x}_k$
3: Create sign matrix S with each $s_{ij} = \text{sign}(w_{ij})$
4: **Do**
5: $c_j := \frac{\sum_{s_{ij} \neq 0} \text{abs}(w_{ij})}{\sum_i s_{ij}^2}$
6: **for** each $s_{ij} \neq 0$ **do**
7: **if** $c_j \geq 2 \cdot \text{abs}(w_{ij})$ **then**
8: $s_{ij} := 0$
9: Remove x_i or $\neg \mathsf{x}_i$ from rule r_j
10: **end if**
11: **end for**
12: **Until** the value of c_j is unchanged
13: **end for**

TABLE III
TRUTH TABLE FOR XOR FUNCTION

x_1	x_2	x_3
$false$	$false$	$false$
$false$	$true$	$true$
$true$	$false$	$true$
$true$	$true$	$false$

The intention here is that for each rule j, an input x_i with small weight $\text{abs}(w_{ij}) \leq c_j/2$ should not appear in the rule. The rule, therefore, only captures the relations between input variables having *strong connection* through the hidden unit j. From (10) and (12), the following extraction algorithm is derived.

Suppose an RBM has I visible units and J hidden units. Rule extraction Algorithm 2 has worst case time complexity $\mathcal{O}(I \times J)$. This is an improvement on searching through the combinations of input vectors, which have worst case time complexity $\mathcal{O}(2^I)$.

1) Working Example (XOR Problem): In this example, we show how confidence rules can be extracted from an RBM trained on the XOR function shown in Table III. An RBM with visible units x_1, x_2, x_3 and ten hidden units was trained to learn this truth table using input value 0 to denote the truth value *false* and input value 1 to denote *true*.

In this example, ten rules exist with antecedents x_1, x_2, and x_3, and consequent h_i, $1 \leq i \leq 10$. The rules extracted from the trained RBM are shown in Table IV. Assuming that x_3 is a target proposition, one can see by inspecting Table IV that among the rules with the higher confidence values, x_3 is associated with either $\mathsf{x}_1 \wedge \neg \mathsf{x}_2$ or $\neg \mathsf{x}_1 \wedge \mathsf{x}_2$. Similarly, $\neg \mathsf{x}_3$ is associated with $\neg \mathsf{x}_1 \wedge \neg \mathsf{x}_2$ or with $\mathsf{x}_1 \wedge \mathsf{x}_2$, as expected. In this case, a rule such as $7.355 : \mathsf{h}_8 \leftrightarrow \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg \mathsf{x}_3$ can be read as: if x_1 is true and x_2 is true, then assuming that h_8 is true, z should be false with confidence 7.355. Observe how in this way any subset of the visible units could have been assumed to be the target proposition.

Given a trained RBM, it is clear that Algorithm 2 would also apply to any subset I of the RBM's visible units.

TABLE IV
RULES EXTRACTED FROM RBM TRAINED ON XOR

$1.340 : \mathsf{h}_1 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \mathsf{x}_3$	$1.677 : \mathsf{h}_6 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \mathsf{x}_3$
$2.970 : \mathsf{h}_2 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \mathsf{x}_3$	$2.544 : \mathsf{h}_7 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \mathsf{x}_3$
$6.165 : \mathsf{h}_3 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \mathsf{x}_3$	$7.355 : \mathsf{h}_8 \leftrightarrow \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg\mathsf{x}_3$
$0.158 : \mathsf{h}_4 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg\mathsf{x}_3$	$6.540 : \mathsf{h}_9 \leftrightarrow \neg\mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \neg\mathsf{x}_3$
$2.481 : \mathsf{h}_5 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \mathsf{x}_3$	$4.868 : \mathsf{h}_{10} \leftrightarrow \neg\mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \neg\mathsf{x}_3$

TABLE V
RULES FROM PART OF THE RBM TRAINED ON XOR

$1.499 : \mathsf{h}_1 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2$	$1.908 : \mathsf{h}_6 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2$
$2.782 : \mathsf{h}_2 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2$	$2.638 : \mathsf{h}_7 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2$
$6.134 : \mathsf{h}_3 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2$	$7.375 : \mathsf{h}_8 \leftrightarrow \mathsf{x}_1 \wedge \mathsf{x}_2$
$0.139 : \mathsf{h}_4 \leftrightarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2$	$6.433 : \mathsf{h}_9 \leftrightarrow \neg\mathsf{x}_1 \wedge \neg\mathsf{x}_2$
$2.582 : \mathsf{h}_5 \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2$	$4.823 : \mathsf{h}_{10} \leftrightarrow \neg\mathsf{x}_1 \wedge \neg\mathsf{x}_2$

$$5.720 : \mathsf{x}_3 \leftrightarrow \mathsf{h}_1 \wedge \mathsf{h}_2 \wedge \mathsf{h}_3 \wedge \neg\mathsf{h}_4 \wedge \mathsf{h}_5 \wedge \mathsf{h}_6 \wedge \mathsf{h}_7 \wedge \neg\mathsf{h}_8 \wedge \neg\mathsf{h}_9 \wedge \neg\mathsf{h}_{10}$$

The algorithm could equally be applied from the hidden units to the remaining visible units not in I. In this way, an RBM where the visible units have been split into input and target units can be seen as a single-hidden-layer DBN to which Algorithm 2 is applied twice for the sake of rule extraction. This idea will be explored in detail in the next section. Returning to the XOR example, assuming $I = \{\mathsf{x}_1, \mathsf{x}_2\}$, the rules in Table V can be extracted.

From the above rules, an assignment of truth values to x_1 and x_2 allows us to derive confidence values for x_3 by applying inference rule INF1. In this example, as expected, x_3 will have a high confidence value when either x_1 is $true$ or x_2 is $false$ or vice versa, but not when both x_1 and x_2 are assigned the same truth value.

Observe that if either x_1 or x_2 was chosen as the target variable, the above same procedure could be applied, without the need for retraining the RBM. Differently from extraction from supervised models [8], [9], here the target does not have to be chosen in advance and the model retrained for each target.

2) Low-Cost Representation: In RBMs, the state of the hidden units for the given state of the visible units can be used as latent features, which, in many cases, can improve the training of a classifier. Frequently, the accuracy of, say, a support vector machine (SVM) will improve when it is trained having latent features such as the state of an RBM's hidden units as input, instead of raw data (that is, the state of the RBM's visible units). Let us investigate whether the same holds for confidence rules. Given one's confidence in a set of beliefs, the confidences of the hypotheses referred to by the confidence rules can be inferred using rule of inference INF1. These confidence values can also be used as input to a classifier. In what follows, we compare the performance of RBMs with that of confidence rules on large image data sets. The confidence rules are shown to offer a more compact representation than RBMs.

We compare the performances of the confidence rules and the latent features of the RBM using three data sets: the TiCC handwritten characters, the MNIST handwritten digits, and the Yale faces data set. All three data sets consist of images with pixel values ranging from 0 to 255, which are then normalized onto the interval [0-1]. Hence, for the purpose of carrying out inference using the rules, we convert each pixel value $x_i = \alpha_i$ into a proposition x_i having confidence value α_i. The TiCC data set consists of 18 189 training samples, 1250 validation samples, and 18 177 test samples. MNIST consists of 60000 training and 10000 test samples. The Yale data set contains 135 training and 30 test samples. Model selection was performed by running a grid search (except for the Yale data set) over the learning rates for the RBMs (between 0.001 and 1) and for the SVM, cost (between 0.0001 and 100), and gamma (between 0.0001 and 100), all on a log scale. On the MNIST data set, for efficiency, we have used 10000 samples for training and 2000 samples for validation. Once the best hyperparameters were selected, the entire set of 60000 samples was used for training the best model, as is normally the case, which was then tested on the unseen 10000 test samples. The number of hidden units was fixed at 500 (we have also trained an RBM with 1000 hidden units, which produced a nonstatistically significant improvement in performance). The classifier used was an SVM with Gaussian kernel [38].

Table VI contains the test set accuracies of the SVMs trained using as input the values of the latent variables from the RBMs with 500 and 1000 hidden units, trained on the three data sets, compared with the accuracies of the SVMs trained using as input the confidence values obtained by applying rule of inference INF1 on the rules extracted by Algorithm 2 from the same RBMs. Each experiment was run ten times and we report the mean accuracies on the hold-out test sets, along with the standard deviation. The results show that the performance of the rules can be (consistently) almost identical to that of the RBMs. In addition, in the rule set, each (positive or negative) literal can be represented by 1 b, while in the RBM, each weight value must be represented by a double data type (i.e., 64 b in a 32-b computer). Therefore, the rules require far less storage memory than the RBMs while still preserving similar performance. Confidence rules can be seen, therefore, as a suitable low-cost representation for RBMs.

B. Knowledge Extraction From DBNs

Following a layerwise approach [3], [15], a hierarchy of confidence rules can be built for the extraction of rules from DBNs through the repeated application of Algorithm 2. Let us start by considering in more detail the case discussed earlier of a single-hidden-layer DBN created by splitting the visible layer of an RBM into input and target subsets and applying Algorithm 2 twice. This will be followed by the presentation of the general-case algorithm for rule extraction from DBNs.

1) Working Example (DNA Promoter Problem): The DNA promoter data set [8] has 106 examples, each consisting of a sequence of 57 nucleotides (A, T, G, or C) from position -50 to $+7$ in the DNA; 53 examples are gene promoters and 53 examples are not. Let us use n_p to denote a nucleotide n at position p, such that $\mathsf{a}_p, \mathsf{t}_p, \mathsf{g}_p, \mathsf{c}_p$ indicate, respectively, that $n_p = A, n_p = T, n_p = G$, and $n_p = C$.

For each variable n_p, a group of four visible units was created in the RBM. Two target units were also added, one

TABLE VI

ACCURACY OF SVMS TRAINED ON THE TICC, MNIST, AND YALE DATA SETS USING THE HIDDEN FEATURES OF AN RBM (WITH EITHER 500 OR 1000 HIDDEN UNITS) AS INPUT, COMPARED WITH THE ACCURACY OF SVMS TRAINED USING AS INPUT THE CONFIDENCE VALUES OF THE HYPOTHESES DERIVED FROM THE RULES EXTRACTED FROM THE RBMS (SEE ALGORITHM 2) BY APPLYING INFERENCE RULE INF1

	TiCC SVM	MNIST SVM	Yale SVM
RBM (J=500)	$94.851\% \pm 0.033$	$98.553\% \pm 0.031$	$95.00\% \pm 2.833$
Confidence rules	$94.711\% \pm 0.072$	$98.530\% \pm 0.040$	$94.333\% \pm 3.865$
RBM (J=1000)	$94.928\% \pm 0.016$	$98.680\% \pm 0.024$	$97.000\% \pm 2.919$
Confidence rules	$94.729\% \pm 0.070$	$98.562\% \pm 0.035$	$96.667\% \pm 1.757$

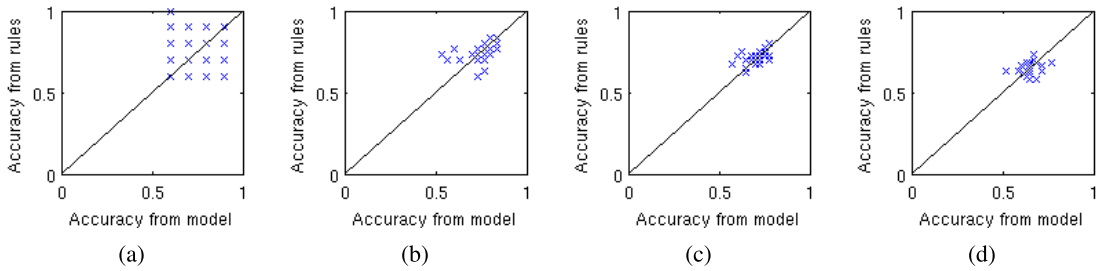

Fig. 1. Classification performances of RBMs and the extracted rules on DNA promoter data set. (a) 96 training and 10 test. (b) 76 training and 30 test. (c) 66 training and 40 test. (d) 46 training and 60 test.

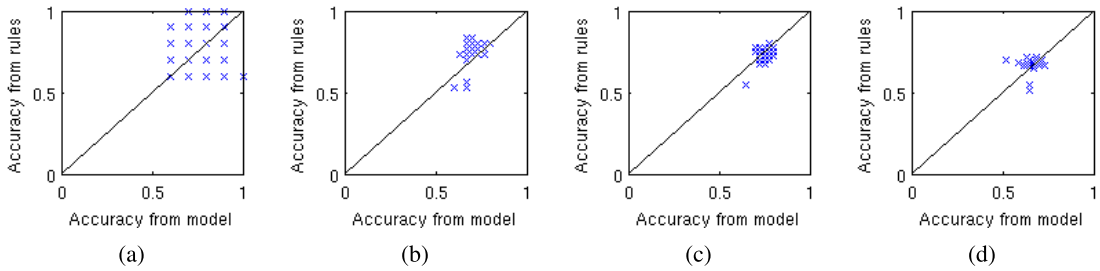

Fig. 2. Classification performances of DBNs compared with extracted rules on the DNA promoter data set. (a) 96 training and 10 test. (b) 76 training and 30 test. (c) 66 training and 40 test. (d) 46 training and 60 test.

for promoter and one for ¬promoter. Five RBMs were trained using 96 examples, each with ten examples randomly selected from the original 106 for testing. Only three hidden units were used (h_1, h_2, h_3); this was sufficient for the networks to achieve 90.67% training set performance on average. By applying inference rule INF1 on these rules, all ten test examples were classified correctly, with promoter obtaining a higher confidence value than ¬promoter whenever the DNA sequence was a gene promoter. On average, for the five RBMs, the rules extracted (such as the two rules that follow) have achieved a test set accuracy of $90\% \pm 6.9296$:

$$0.761 : \neg\text{promoter} \leftrightarrow \neg h_1^{(1)} \wedge h_2^{(1)} \wedge \neg h_3^{(1)}$$
$$1.042 : \text{promoter} \leftrightarrow h_1^{(1)} \wedge h_3^{(1)}.$$

Direct comparisons with other extraction approaches such as MofN [8] and RuleSet [9] would be nontrivial because of the differences in methodology and learning method (supervised versus unsupervised). Nevertheless, for completeness, we report here the results obtained by those extraction methods on the DNA promoter problem. The MofN approach is reported to have achieved 92.5% accuracy using tenfold cross validation, while RuleSet achieved nine correct classifications out of ten test set examples on a rule set extracted from a feedforward neural network trained using backpropagation on the remaining 96 examples.

Exploring the DNA promoter experiment more systematically, let us now evaluate empirically the impact of the information loss expected as part of the process of rule extraction. In order to do this, in what follows, we compare the test set accuracy of the rules extracted from the DBN with that of the DBN itself. This evaluation was done for four different partitions of training and test data, as shown in Figs. 1 and 2. For each partition, 20 networks were trained using different settings. Figs. 1 and 2 plot the classification performance of the network model against that obtained by the corresponding rule set. The results indicate a high fidelity of the rules.[1]

We now turn our attention to the special nature of certain nodes in the network, as seen in the case of the group of four in the DNA promoter problem and notably when the target nodes are expected to be exclusive (e.g., as part of a softmax target layer in the network). In this case, the rules extracted

[1]Observe that the grid-like arrangement of accuracies in Figs. 1(a) and 2(a) is a result of the small number of test samples in such cases (ten samples only).

Algorithm 3 TOP_RBM_EXTRACT

Require: An RBM with visible layer I, hidden layer H, and label layer Y
1: R = ∅, T = ∅
2: R = RBM_EXTRACT(I, H)
3: **for** each hidden unit $j \in$ H and output unit $o \in$ Y **do**
4: Add a rule : $e^{u_{oj}} : \mathbf{y} = o \leftrightarrow \mathsf{h}_j$ to T
5: **end for**
6: **return** R, T

are expected to follow the conditional distribution:

$$P(\mathbf{y} = o|\mathbf{x}) \propto \prod_j (1 + e^{\sum_i w_{ij} x_i + u_{oj} y_o}) \tag{13}$$

where U is the weight matrix between the label layer Y and the hidden layer H and $\mathbf{y}$ is a one-hot vector representing the label classes. For example, $\mathbf{y} = o$ represents class o, where $y_o = 1$ and $y_{o' \neq o} = 0$. From (13), the following confidence function can be defined:

$$\mathcal{C}(\mathbf{y} = o, \mathbf{x}) = \sum_j \log(1 + e^{\sum_i w_{ij} x_i} e^{u_{oj} y_o}). \tag{14}$$

Algorithm 2 accounts for the first product in the above equation by extracting rules from input I to the hidden layer $\mathsf{h}_j^{(1)}$ (or, more generally, from $\mathsf{h}_j^{(i)}$ to $\mathsf{h}_j^{(i+1)}$). With $\mathbf{y} = o$ expressed as $y_o = 1$, the exponential in the second product in the above equation can be used to normalize the confidence values $\alpha_j^{(1)}$ of $\mathsf{h}_j^{(1)}$, producing

$$\mathcal{C}(\mathbf{y} = o, \mathbf{x}) \approx \sum_j \log(1 + \alpha_j^{(1)} e^{u_{oj}}). \tag{15}$$

Given (15), for each hidden unit and hypothesis $\mathbf{y} = o$, one can extract a rule $e^{u_{oj}} : \mathbf{y} = o \leftrightarrow \mathsf{h}_j^{(1)}$, whose confidence value is $e^{u_{oj}}$. By applying inference rule INF1 and adding the confidence values of each hypothesis normalized by $f(\alpha) = \log(1+\alpha)$, one obtains the same confidence values as produced by (15). Algorithm 3 formalizes the resulting rule extraction for such softmax layers.

We are now in position to introduce the general algorithm for rule extraction from DBNs (Algorithm 4). It follows a layerwise approach, whereby for a DBN having n layers, either Algorithm 2 is applied n times or Algorithm 2 is applied $n-1$ times and Algorithm 3 is applied once. We call the first alternative *compact* as it generates a fewer rules at the top level of the DBN.

2) Information Loss in Complex Domains: We now test the general method of rule extraction from DBNs (Algorithm 4) on a harder problem, namely, the MNIST handwritten digit recognition data set. This is a difficult problem for rule extraction because the inputs are the values of the pixels in the images to be classified into classes 0, 1, 2, . . . , 9. The rules are therefore expected to capture the levels of abstraction learned by the DBN, from the raw data through to the class, hopefully identifying useful concepts such as edges and shapes as part of the rule hierarchy. Such image domains are notoriously difficult for symbolic reasoning.

Algorithm 4 DBN_EXTRACT

Require: A stack of L RBMs
1: Create empty rule set R = ∅
2: **for** $l = 1$ to $L - 1$ **do**
3: $\mathrm{R}^{(l)}$ = RBM_EXTRACT(l, l+1)
4: Add $\mathrm{R}^{(l)}$ to R
5: **end for**
6: **if** COMPACT **then**
7: $\mathrm{R}^{(L)}$ = RBM_EXTRACT(L, Y)
8: **else**
9: $\mathrm{R}^{(L)}$ = TOP_RBM_EXTRACT(L, Y)
10: **end if**
11: Add $\mathrm{R}^{(L)}$ to R

In what follows, we report results using *COMPACT* = *False* (see Algorithm 4). In this image domain, we found that information loss is larger when *COMPACT* = *True*. We attribute information loss in this case of the MNIST data set to the fact that the input data is nonbinary, showing more variance than the DNA data evaluated earlier. As a result, in the case of a deep network, information loss may be compounded when inference is applied sequentially through the rule hierarchy (i.e., without sampling). In what follows, we evaluate information loss in more detail.

In Section IV-A2, the confidence values of the rules extracted from an RBM were provided as input for training an SVM. In the case of a DBN, the same layerwise approach would result in each RBM in the hierarchy being trained and the rules extracted before the next RBM can be trained. In order to evaluate information loss in DBNs, though, instead of doing the above, we are interested in the extraction of a complete hierarchy of rules from the entire DBN. We have trained 155 DBNs using the standard configuration used by others [3]: 784 input nodes, 500 nodes in the first hidden layer, 1000 nodes in the second hidden layer, and 10 target nodes. We have used the benchmark MNIST data set with 20 000 training examples, 10 000 held-out examples used for early stopping validation [39], and 10 000 test examples. Fig. 3 shows for the MNIST data, as done for the DNA promoter data, a comparison between the test set accuracy of the DBNs (model accuracy) and the test set accuracy of the extracted rules. As expected, the results indicate more information loss here than in the case of the DNA data, with an average information loss in the rules of 15.3026% ±5.9255 in relation to the DBNs.

The DBNs were trained using learning rate decay and early stopping based on their performance on the validation set (whenever the validation set error increased, a lower learning rate was used for network training). The same can be done using rule sets extracted from the network as follows: rules are extracted after each epoch of training. Instead of the network, the rules are used to calculate the validation set error. Whenever the validation error increases using the rules, a lower learning rate is used in the training of the network. In this way, the extracted rules are used to trigger the early stopping of the network training. It is expected that using the

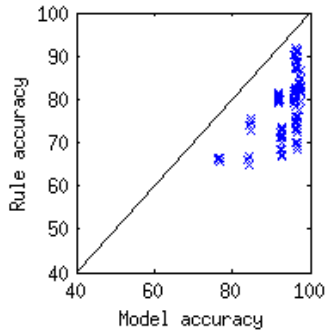

Fig. 3. Comparison between the test set accuracies of DBNs (model accuracy) and extracted rules on the MNIST data set.

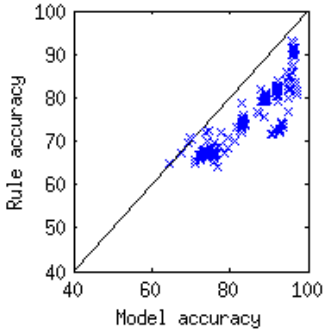

Fig. 4. Comparison between the test set accuracies of DBNs (model accuracy) and extracted rules using rule-based early stopping on the MNIST data set.

rules for validation, the fidelity of the rules to the final model should increase. Nevertheless, since the time complexity of the extraction of rules from each RBM is $\mathcal{O}(I \times J)$, the computation cost for validation by rules is higher than for validation using the model itself. Fig. 4 shows a comparison between the test set accuracy of the DBNs (model accuracy), now using such rule-based early stopping, and the test set accuracy of the extracted rules. Now, an average information loss of 9.2519% $\pm$4.2095 is achieved in relation to the DBNs. In comparison with Fig. 3, it can be seen that the use of rule-based early stopping produces rule sets with higher fidelity to the network model (i.e., lower information loss). Given the complexity of image domains when it comes to rule extraction, we interpret the results shown in Fig. 4 as indicative that the extracted rules can be useful at highlighting certain important relationships in the network models, e.g., if the same or very similar rules are extracted from the various network models. This domain specific analysis is left as future work.

Achieving a higher level of integration between network and rule models at learning may be desirable, as seen, e.g., above when extracted rules were used as a criterion for the network's early stopping. Such integration can be achieved fully through the provision of algorithms for inserting rules into network models. This will be the topic of discussion for the rest of this paper.

V. Deep Neural–Symbolic Integration

Having seen how symbolic knowledge can be extracted from DBNs, we now investigate the inverse problem of inserting

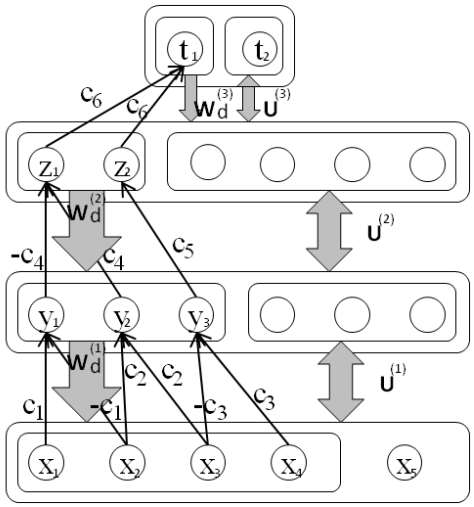

Fig. 5. DBN obtained from hierarchical rule set K from Example 2.

symbolic knowledge into DBNs to improve network learning using background knowledge. The idea of encoding knowledge into DBNs to improve learning performance is inspired by early work on knowledge-based neural networks [34], [35]. In addition to improving learning time, prior knowledge has been shown capable of improving learning accuracy by allowing knowledge that is not reinforced through learning, but that might nevertheless be relevant, to persist in the network model.

A. Knowledge Encoding

In this section, we propose a method and algorithm for encoding confidence rules into DBNs. We also perform an evaluation of knowledge insertion using both the DNA and MNIST data sets used earlier. This evaluation shows that as expected, improvements in performance can be achieved with the use of prior knowledge. We argue, therefore, that when prior knowledge is available, the provision of algorithms allowing its use within network models (such as the algorithm introduced in this section) is desirable.

As has been discussed in Section IV, a hierarchical knowledge base with associated confidence values can offer an appropriate symbolic representation for DBNs. In fact, such a representation has been motivated by the way that DBNs work, as indicated by the way that a hierarchical weighted knowledge base has been defined (see Definition 1).

Example 2 and Fig. 5 illustrate the main idea behind the encoding algorithm to follow using a simple set of rules. Fig. 5 also illustrates how the DBN can be extended to account for learning from data and background knowledge, which is discussed in the sequel.

Example 2: (Encoding Knowledge) Given a hierarchical set of rules $K^s = \{K^{(1)}, K^{(2)}, K^{(3)}\}$, where

$$
\begin{aligned}
K^{(1)} &= \{c_1 : \mathsf{y}_1 \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2;\ c_2 : \mathsf{y}_2 \leftrightarrow \mathsf{x}_2 \\
&\qquad \wedge \mathsf{x}_3;\ c_3 : \mathsf{y}_3 \leftrightarrow \neg \mathsf{x}_3 \wedge \mathsf{x}_4\} \\
K^{(2)} &= \{c_4 : \mathsf{z}_1 \leftrightarrow \neg \mathsf{y}_1 \wedge \mathsf{y}_2;\ c_5 : \mathsf{z}_2 \leftrightarrow \mathsf{y}_3\} \\
K^{(3)} &= \{c_6 : \mathsf{t}_1 \leftrightarrow \mathsf{z}_1 \wedge \mathsf{z}_2\}.
\end{aligned}
$$

Algorithm 5 Rule Encoding Algorithm

Require: a hierarchical weighted knowledge-base K

for $l = 1$ to L **do**
  Initialize an empty RBM $N^{(l)}$;
  for each rule $c_j^{(l)} : \mathsf{h}_j^{(l)} \leftrightarrow \bigwedge_t \mathsf{h}_t^{(l-1)} \wedge \bigwedge \neg \mathsf{h}_k^{l-1)} \in K_l$ **do**;
    Add a unit j to hidden layer l;
    Set the value of the connection weight w_{tj}^l from node $\mathsf{h}_t^{(l-1)}$ to node j to c_i;
    Set the value of the connection weight w_{kj}^l from node $\mathsf{h}_k^{(l-1)}$ to node j to $-c_i$;
  end for
  if $l > 1$ **then**
    Stack $N^{(l)}$ on top of $N^{(l-1)}$;
  end if
end for

For a data set with variables $\{x_1, x_2, x_3, x_4, x_5, t_1, t_2\}$, consider the rule $c_1 : \mathsf{y}_1 \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2$. We add a unit y_1 to the hidden layer of the first RBM and set the weights to $w_{11} = c_1, w_{21} = -c_1$. We repeat the process for each rule in $K^{(1)}$ and create random down-weight connections for the units. We then repeat the process for each level of the hierarchy. Finally, we allow the addition of extra hidden nodes with bidirectional random connections to each hidden level. Fig. 5 shows the resulting network for hierarchical set K.

Algorithm 5 shows how a hierarchical weighted knowledge base can be encoded into a DBN. Since the connections in an RBM are bidirectional, while the rules support only bottom-up inference, the confidence values are encoded as *up-weights* in the network, with a set of down-weights with random values being added from the hidden units to the visible units.

B. Learning With Background Knowledge

Let $\mathcal{K}$ be a hierarchical weighted knowledge base, that is, a hierarchical set of implication rules with *confidence values*, as defined earlier. We have encoded each subset of rules $\mathcal{K}^{(l)}$ at each level of the hierarchy into an RBM and have added more hidden units to it (the number of extra hidden units to add will be investigated empirically). For each RBM, the energy function is

$$\mathbf{E}(\mathbf{x}, \mathbf{h}) = -\sum_j h_j c_j \sum_i s_{ij} x_i - \sum_{ik} x_i u_{ik} h_k - \sum_i a_i x_i - \sum_k b_k h_k \tag{16}$$

where $\mathbf{x}$ and $\mathbf{h}$ denote units added by Algorithm 5, associated with background knowledge rules, c_j is the initial *confidence value* of rule j, J denotes the number of rule-encoded units in the hidden layer (corresponding to the number of rules), K is the number of extra units added to the hidden layer, $s_{ij} = 1$ if the encoded weight is positive, $s_{ij} = -1$ if the encoded weight is negative, or $s_{ij} = 0$ if the weight is zero (see Algorithm 5), and $u_{ij} \in U$ is the value of the weights of the extra hidden units (see Fig. 5).

Algorithm 6 Learning With Guidance

Require: A set of rules $K_l^{(s)}$; input data X

Select a number of rules in $K_l^{(s)}$ with the highest confidence values
Encode $K_l^{(s)}$ in hidden units $H^{(s)}$ with up-weights $W_u^{(l)}$ and random down-weights $W_d^{(l)}$
Add extra hidden units $H^{(t)}$ with weights $U^{(l)}$
repeat
% positive stage: assign X to visible layer
  $X_{pos} := X$;
  $H_{pos} := P(H|X_{pos})$; $\quad \hat{H}_{pos} \sim P(H|X_{pos})$;
  $X_{neg} := P(X|\hat{H}_{pos})$; $\quad \hat{X}_{neg} \sim P(X|\hat{H}_{pos})$;
% negative stage
  $H_{neg} = P(H|\hat{X}_{neg})$;
  $W_d^{(l)} = W_d^{(l)} + \eta(\langle X_{pos}^\top H_{pos}^{(s)} - \hat{X}_{neg}^\top H_{neg}^{(s)} \rangle)$
  $U^{(l)} = U^{(l)} + \eta(\langle X_{pos}^\top H_{pos}^{(t)} - \hat{X}_{neg}^\top H_{neg}^{(t)} \rangle)$
until convergence

The encoded knowledge will be used to guide learning within the neural–symbolic RBMs by maximizing the log-likelihood of the parameters for the given data and background knowledge. Since the connections in an RBM are bidirectional, while the background knowledge supports only bottom-up inference, we split the connection weights between visible and rule-encoded hidden units. The confidence values were used to define the *up-weights* (W_u) and random values were assigned to the *down-weights* (W_d). The learning algorithm in the following will, therefore, adapt the parameters that consist of additional connection weights U and the down-weights W_d for the given confidence values.

We use contrastive divergence [26] to train the networks. The log-likelihood function is given by

$$\mathcal{L}_{lRBM} = \sum_{\mathbf{x} \in \mathcal{D}} P(\mathbf{x}|\theta = \{U, W_d\}; \mathcal{K}). \tag{17}$$

The gradient of the log-likelihood function is given by

$$\Delta u_{ik} = \langle x_i h_k \rangle_0 - \langle x_i h_k \rangle_{\mathcal{K}} \tag{18}$$

where $\langle . \rangle_{\mathcal{K}}$ is the average over the set of examples for a given neural–symbolic network with background knowledge $\mathcal{K}$. We call the following learning algorithm *learning with guidance* because prior knowledge is used to partially fix some upward connections in the network; all other connections, both downward and bidirectional, are allowed to change as part of learning using standard contrastive divergence.

C. Experiments on DNA Promoter Data Set

In this experiment, we use the domain theory provided with the DNA promoter data set[2] to set up and train a DBN. As before, we use variable n_p to denote a nucleotide at position p such that, e.g., a_p represents $n_p = A$. Hence, the background knowledge rule $minus_{10} \leftrightarrow n_{-12} = T \wedge n_{-11} = A \wedge n_{-7} = T$ becomes $c : \mathsf{minus}_{10} \leftrightarrow \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{t}_{-7}$, with arbitrary confidence value c.

[2] http://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Promoter+Gene+Sequences).

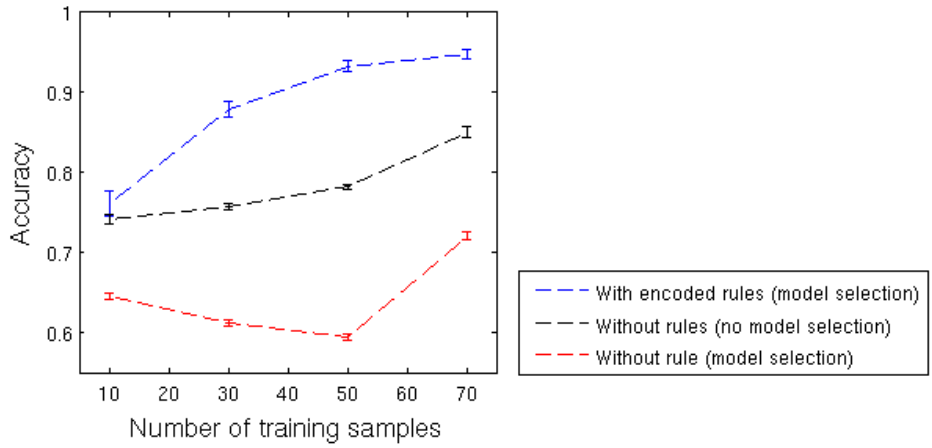

Fig. 6. Test set classification performance of a two-layer standard DBN (red and black lines, with and without model selection) and a two-layer DBN encoded with prior knowledge from the DNA promoter domain theory using training sets with 10, 30, 50, and 70 samples and 20 test samples.

We use Algorithm 5 to encode the domain theory into the network and use Algorithm 6 to train each layer greedily as done in standard DBNs. The domain theory is divided into a hierarchical set of rules that is encoded into a two-layer DBN. We choose two hidden layers because the rule promoter $\leftrightarrow$ contact $\wedge$ conformation consists of variable conformation at level 2 and contact at level 3 of the hierarchy. By combining such rules, we do not need to create another layer with more intermediate variables.[3]

For evaluation, we partition the data into training, validation, and test sets. In order to investigate how background knowledge influences learning given different amounts of data, we use different training sets with 10, 30, 50, and 70 samples. The validation and test sets are the same for each of the training sets and consist of 16 and 20 samples, respectively. From this experiment, we observe that when using the validation set to select the DBNs without background knowledge, this results in low performance on the test set. For the DBNs encoded with background knowledge, the models selected by the validation set produce good accuracy on the test set. This might happen because the number of hyperparameters is large compared with that of the small validation and test sets. Hence, the grid search tends to produce a DBN that overfits the validation set. When background knowledge is encoded into the networks, overfitting is avoided through the use of the rule guidance learning algorithm, which seems to be able to produce a more general model.

In order to make a better comparison between the standard DBN and the encoded DBN, we include in Fig. 6 the best accuracy achieved on the test set using the standard DBN without using model selection. Fig. 6 shows that with background knowledge to guide the learning, the DBN can achieve a considerable improvement in performance (e.g., up by 15% when the training set has size 50). It also shows that when the training set is larger (70 examples), the standard DBN is able to learn from the data alone the knowledge that had been provided as background theory, and therefore, the improvement with prior knowledge is smaller for larger training sets.

[3]We merge rules c : promoter $\leftrightarrow$ contact $\wedge$ conformation and c : contact $\leftrightarrow$ minus$_{35}$ $\wedge$ minus$_{10}$ into c : promoter $\leftrightarrow$ minus$_{35}$ $\wedge$ minus$_{10}$ $\wedge$ conformation.

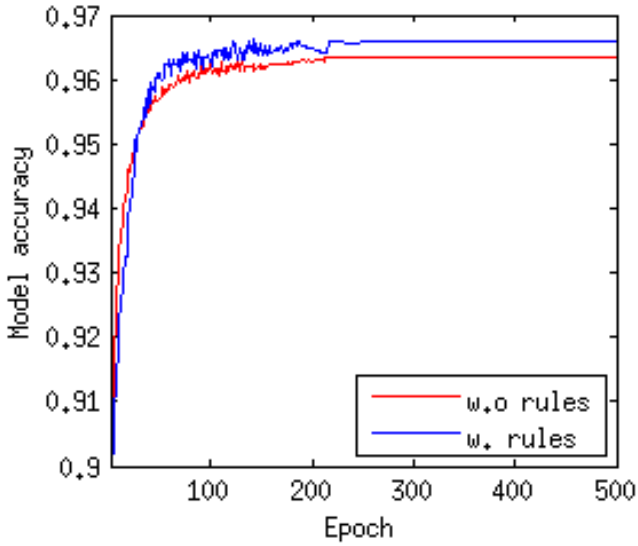

Fig. 7. Test set performance comparison between DBNs trained with and without prior encoding of rules on the MNIST data set; DBNs with rule encoding achieve slightly higher accuracy faster.

D. Experiments on MNIST Data Set

In order to evaluate the influence of background knowledge on the MNIST data set (for which no prior symbolic knowledge is available), we have extracted rules from a network trained on a subset of the data by applying Algorithm 4 and then inserted such rules into a new network for further training and comparison. In the MNIST data set, there were 20 000 examples for training, 10 000 examples for validation, and 10 000 examples for testing. We have selected 1000 examples randomly from the training set for training a two-layer DBN from which rules were then extracted. Such rules were encoded into a new DBN, following the procedure in Section V-A. This new DBN was then trained on the

remaining 19 000 examples. Finally, results were compared with those obtained by another DBN trained from scratch on the entire 20 000 examples without any rule insertion.

Fig. 7 shows that with the encoding of rules, a DBN can achieve a slightly higher accuracy faster than a DBN without rules. This suggests that the network structure may be important. Although there is information loss within the rules extracted from the DBN trained on the 1000 examples, the DBN setup with such rules and trained on the remaining 19 000 examples performed slightly better than the DBN trained on the entire 20 000 examples in one goal, which included those 1000 examples. This indicates that maintaining the structures of the representations learned by deep networks can be beneficial as part of a modular approach to learning. This requires further research.

VI. Conclusion

This paper introduced and evaluated algorithms for inserting and extracting knowledge from deep networks. The question of whether modularity can help the integration of learning and reasoning in deep networks has been investigated empirically. A new inference rule for deep networks using *confidence rules* has been proposed, which combines symbolic representation and quantitative reasoning. This inference rule and logical representation was designed to support hierarchical reasoning and was shown to be an adequate representation for the modular training of deep networks. Knowledge represented by confidence rules can be inserted or extracted from DBNs. It is shown that in single-layer DBNs, also known as RBMs, confidence rules offer a low-cost representation for the RBMs. The use of a modular layerwise approach to knowledge extraction from DBNs is shown to produce information loss at times, notably when the networks are trained on complex image data. Yet, the modular training of networks as part of a cycle of knowledge insertion, learning, and extraction can produce an improvement in performance (see Figs. 6 and 7). Knowledge encoding into DBNs in the form of confidence rules has been shown to be useful, leading to an improvement in performance following a layerwise training. The results from this work suggest that there is promise in building a hierarchical reasoning system capable of integrating symbolic and subsymbolic capabilities. With the layerwise extraction and insertion principle, the work in this paper can be extended to deeper networks. However, for knowledge extraction from very deep networks, as discussed earlier, the use of extracted rules for model selection may not be practical due to an increased computational overhead.

As future work, further domain specific experimental evaluations may be carried out. Of particular interest are applications of multimodal data using very deep networks. Recent research shows a deep semantic mapping between text and images [40], which motivates the use of knowledge extraction from, say, a text modality as relevant for the explanation of context in the image modality. Another direction for future work includes the parallel implementation of the proposed knowledge insertion and extraction algorithms and its use in the iterative evaluation of very large networks, as part of a cycle of knowledge acquisition and revision. In this process, one may consider the use of recent techniques from high-performance computing, which, applied to neural networks, may take advantage of the underlying graph structures to achieve large-scale improvements in performance. These include the local parallel method applied to directed acyclic graphs and the multi-index chained hashing method for fast neuron memory search [41]–[43].

References

[1] L. G. Valiant, "Three problems in computer science," *J. ACM*, vol. 50, no. 1, pp. 96–99, Jan. 2003.

[2] J. Y. Halpern, *Reasoning About Uncertainty*. Cambridge, MA, USA: MIT Press, 2003. [Online]. Available: http://opac.inria.fr/record=b1100784

[3] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[4] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing: Foundations*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281.

[5] L. Breiman, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Statist. Sci.*, vol. 16, no. 3, pp. 199–231, Aug. 2001. [Online]. Available: http://dx.doi.org/10.1214/ss/1009213726

[6] A. S. d'Avila Garcez, L. Lamb, and D. M. Gabbay, *Neural-Symbolic Cognitive Reasoning* (Cognitive Technologies). Berlin, Germany: Springer, 2009.

[7] A. S. d'Avila Garcez *et al.*, "Neural-symbolic learning and reasoning: Contributions and challenges," in *Proc. AAAI Spring Symp. Knowl. Represent. Reason., Integr. Symbolic Neural Approaches*, Mar. 2015, pp. 1–4.

[8] G. G. Towell and J. W. Shavlik, "The extraction of refined rules from knowledge-based neural networks," *Mach. Learn.*, vol. 13, no. 1, pp. 71–101, Oct. 1993.

[9] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay, "Symbolic knowledge extraction from trained neural networks: A sound approach," *Artif. Intell.*, vol. 125, nos. 1–2, pp. 155–207, Jan. 2001.

[10] M. G. Augasta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," *Neural Process. Lett.*, vol. 35, no. 2, pp. 131–150, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1007/s11063-011-9207-8

[11] K. Odajima, Y. Hayashi, G. Tianxia, and R. Setiono, "Greedy rule generation from discrete data and its use in neural network rule extraction," *Neural Netw.*, vol. 21, no. 7, pp. 1020–1028, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2008.01.003

[12] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 299–307, Feb. 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TNN.2007.908641

[13] A. Hara and Y. Hayashi, "Ensemble neural network rule extraction using Re-RX algorithm," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Brisbane, QLD, Australia, Jun. 2012, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2012.6252446

[14] T. A. Etchells and P. J. G. Lisboa, "Orthogonal search-based rule extraction (OSRE) for trained neural networks: A practical and efficient approach," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 374–384, Mar. 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TNN.2005.863472

[15] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, vol. 19, B. Scholkopf, J. Platt, and T. Hoffman, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 153–160.

[16] L. Bottou, "From machine learning to machine reasoning," *Mach. Learn.*, vol. 94, no. 2, pp. 133–149, Jan. 2014. [Online]. Available: http://leon.bottou.org/papers/bottou-mlj-2013

[17] H. L. H. de Penning, A. S. d'Avila Garcez, L. C. Lamb, and J.-J. C. Meyer, "A neural-symbolic cognitive agent for online learning and reasoning," in *Proc. IJCAI*, 2011, pp. 1653–1658.

[18] S. Tran and A. S. d'Avila Garcez, "Logic extraction from deep belief networks,"in *Proc. ICML Represent. Learn. Workshop*, Edinburgh, Scotland, Jul. 2012, pp. 1–7. [Online]. Available: https://sites.google.com/site/representationworkshopicml2012/schedule

[19] S. N. Tran and A. S. d'Avila Garcez, "Knowledge extraction from deep belief networks for images," in *Proc. IJCAI-Workshop Neural-Symbolic Learn. Reason.*, 2013, pp. 1–6.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[21] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 873–880.
[22] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2009, pp. 609–616.
[23] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.
[24] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2007, pp. 153–160.
[25] R. Salakhutdinov, "Learning in Markov random fields using tempered transitions," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2009, pp. 1598–1606.
[26] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.
[27] R. Salakhutdinov, "Learning deep Boltzmann machines using adaptive MCMC," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 943–950.
[28] R. V. Borges, A. S. d'Avila Garcez, and L. C. Lamb, "Learning and representing temporal knowledge in recurrent networks," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 2409–2421, Dec. 2011.
[29] M. W. Craven and J. W. Shavlik, "Learning symbolic rules using artificial neural networks," in *Proc. Int. Conf. Mach. Learn.*, 1993, pp. 73–80.
[30] A. S. d'Avila Garcez, K. B. Broda, and D. M. Gabbay, *Neural-Symbolic Learning Systems: Foundations and Applications* (Perspectives in Neural Computing). Berlin, Germany: Springer, 2002.
[31] B. Hammer and P. Hitzler, Eds., *Perspectives of Neural-Symbolic Integration*. Berlin, Germany: Springer, 2007.
[32] G. Pinkas, "Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge," *Artif. Intell.*, vol. 77, no. 2, pp. 203–247, Sep. 1995.
[33] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *Advances in Neural Information Processing Systems*, vol. 19. Cambridge, MA, USA: MIT Press, 2007.
[34] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artif. Intell.*, vol. 70, nos. 1–2, pp. 119–165, 1994.
[35] A. S. d'Avila Garcez and G. Zaverucha, "The connectionist inductive learning and logic programming system," *Appl. Intell.*, vol. 11, no. 1, pp. 59–77, Jul. 1999.
[36] M. Richardson and P. Domingos, "Markov logic networks," *Mach. Learn.*, vol. 62, no. 1, pp. 107–136, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1007/s10994-006-5833-1
[37] L. Getoor and T. Ben, "Introduction to statistical relational learning (Adaptive computation and machine learning)," MIT Press, USA, 2007, ISBN. 0262072882.
[38] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge Univ. Press, 2004.
[39] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Construct. Approx.*, vol. 26, no. 2, pp. 289–315, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00365-006-0663-2
[40] W. Liu, T. Mei, Y. Zhang, C. Che, and J. Luo, "Multi-task deep visual-semantic embedding for video thumbnail selection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3707–3715.
[41] A. Forechi *et al.*, "Fat-fast VG-RAM WNN: A high performance approach," *Neurocomputing*, vol. 183, pp. 56–69, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2015.06.104
[42] C. Yan *et al.*, "Efficient parallel framework for HEVC motion estimation on many-core processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 12, pp. 2077–2089, Dec. 2014.
[43] C. Yan *et al.*, "A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors," *IEEE Signal Process. Lett.*, vol. 21, no. 5, pp. 573–576, May 2014.

Son N. Tran is a research fellow at the Commonwealth Scientific and Industrial Research Organisation, Australia. He holds a Ph.D. in Computer Science (2016) from City, University of London, an EU Erasmus Mundus joint MSc in Networks and e-Business Computing, University of Reading, and a BSc and MSc in Electronic Engineering from the Hanoi University of Technology. Tran is the recipient of the Erasmus Mundus certificate of achievement 2010 in recognition of an outstanding academic performance. His current research interests are machine learning and knowledge discovery. He is a member of the Neural-Symbolic Learning and Reasoning Association.

Artur S. d'Avila Garcez, FBCS, is Director of the Research Centre for Machine Learning at City, University of London. He holds a Ph.D. in Computer Science (2000) from Imperial College London. He co-authored two books: Neural-Symbolic Cognitive Reasoning (Springer, 2009) and Neural-Symbolic Learning Systems (Springer, 2002), and more than 150 peer-reviewed publications in Artificial Intelligence, Machine Learning, Neural Computation and Neural-Symbolic Computing. Garcez is president of the Neural-Symbolic Learning and Reasoning Association, and a member of the editorial boards and programme committees of many journals and international conferences. His research has received funding from the Nuffield foundation, the EU, IBM, CNPq and CAPES Brazil, the Daiwa Foundation, the Royal Society, Innovate, ESRC and EPSRC, UK.