REPRESENTATION DECOMPOSITION FOR KNOWLEDGE EXTRACTION AND SHARING USING RESTRICTED BOLTZMANN MACHINES

by

Son Tran

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy, Department of Computer Science, City University London.

CITY UNIVERSITY LONDON

EST 1894

2016

Abstract

Restricted Boltzmann machines (RBMs), with many variations and extensions, are an efficient neural network model that has been applied very successfully recently as a building block for deep networks in diverse areas ranging from language generation to video analysis and speech recognition. Despite their success and the creation of increasingly complex network models and learning algorithms based on RBMs, the question of how knowledge is represented, and could be shared by such networks, has received comparatively little attention. Neural networks are notorious for being difficult to interpret. The area of knowledge extraction addresses this problem by translating network models into symbolic knowledge. Knowledge extraction has been normally applied to feed-forward neural networks trained in supervised fashion using the back-propagation learning algorithm. More recently, research has shown that the use of unsupervised models may improve the performance of network models at learning structures from complex data. In this thesis, we study and evaluate the decomposition of the knowledge encoded by training stacks of RBMs into symbolic knowledge that can offer: (i) a compact representation for recognition tasks; (ii) an intermediate language between hierarchical symbolic knowledge and complex deep networks; (iii) an adaptive transfer learning method for knowledge reuse. These capabilities are the fundamentals of a Learning, Extraction and Sharing (LES) system, which we have developed. In this system *learning* can automate the process of encoding knowledge from data into an RBM, *extraction* then translates the knowledge into symbolic form, and *sharing* allows parts of the knowledge-base to be reused to improve learning in other domains. To this end, in this thesis we introduce *confidence rules*, which are used to allow the combination of symbolic knowledge and quantitative reasoning. Inspired by Penalty Logic - introduced for Hopfield networks confidence rules establish a relationship between logical rules and RBMs. However, instead of representing propositional well-formed formulas, confidence rules are designed to account for the reasoning of a stack of RBMs, to support modular learning and hierarchical inference. This approach shares common objectives with the work on neural-symbolic cognitive agents. We show in both theory and through empirical evaluations that a hierarchical logic program in the form of a set of confidence rules can be constructed by decomposing representations in an RBM or a deep belief network (DBN). This decomposition is at the core of a new knowledge extraction algorithm which is computationally efficient. The extraction algorithm seeks to benefit from the symbolic knowledge representation that it produces in order to improve network initialisation in the case of transfer learning. To this end, confidence rules offer a language for encoding symbolic knowledge into a deep network, resulting, as shown empirically in this thesis, in an improvement in modular learning and reasoning. As far as we know this is the first attempt to extract, encode, and transfer symbolic knowledge among DBNs. In a confidence rule, a real value, named *confidence value*, is associated with a logical implication rule. We show that the logical rules with the highest confidence values can perform similarly to the original networks. We also show that by transferring and encoding representations learned from a domain onto another related or analogous domain, one may improve the performance of representations learned in this other domain. To this end, we introduce a novel algorithm for transfer learning called "Adaptive Profile Transferred Likelihood", which adapts transferred representations to target domain data. This algorithm is shown to be more effective than the simple combination of transferred representations with the representations learned in the target domain. It is also less sensitive to noise and therefore more robust to deal with the problem of negative transfer.

Keywords: Unsupervised Learning, Restricted Boltzmann Machines, Deep Belief Networks, Knowledge Extraction, Neural-symbolic Integration, Transfer Learning.

Acknowledgements

First and foremost I would like to thank my supervisor Artur d'Avila Garcez for supporting me through all the hard work to complete my PhD thesis. The encouragement and inspiration he has brought to me since the first day I came to City University London (the City) have great influence to my work. Artur is not only an experienced supervisor but also my research companion who shares with me the fun and the beauty of AI. He can spend hours to listen to my explanations, most of which I barely understand if I do this to myself.

I am fortunate to have an opportunity to work with many excellent researchers at the City, especially Tillman Weyde, Daniel Wolff and Srikanth Cherla in applying representation learning and deep learning to music informatics. I also enjoyed collaborating with Emmanouil Benetos (QMUL) in the work of action recognition, Thanh Vu (OU) in search personalisation and Hazrat Ali (USTB) in speaker recognition. During my time at City, I have received huge support from the administrative and technical staff, especially Mark Firman and Chris Marshall.

I gratefully acknowledge the funding source that made my Ph.D work possible. For this, I would like to thank Jason Dykes for supporting my application, and after that, my work as the second supervisor. My research was funded by the studentship from City University London. During my PhD years I have received a number of other funds from City to attend conferences and workshops. I also want to thank Chris Child for having me in the pump-priming project. This funding allows me to establish the ideas of future work during the writing up.

For the completion of this thesis I would like to extend the sincerest appreciation to Daniel Silver, Emmanouil Benetos, Srikanth Cherla, Manoel Franca, Andrew

Lambert, Muhammad Asad, Nathan Olliverre, Rilwan Basaru and Hazrat Ali, who have spent their precious time to read and give me valuable comments. I want to send my deep gratitude to Kostas Stathis and Gregory Slabaugh for the fair examination of my thesis and for their constructive suggestions.

Last but not least, I dedicate this work and give the special thanks to my family for their love, support and encouragement. For my parents who always believe in me and my brother Ha who carried out my duties when I was working away. Especially for my wife Chi and my little Hai Dang who are always beside me to make everything I do worth doing.

Contents

List of Figures

List of Tables

Notations

Matrices, vectors are denoted using bold capital ($\mathbf{X}$), boldface ($\mathbf{x}$) letters respectively. A constant is denoted as normal capital letter (N). Italic letters (E, f,g) denote a function. Concatenation of matrices/vectors are denoted as $[\mathbf{X}, \mathbf{Y}]$ for column order and $[\mathbf{X}; \mathbf{Y}]$ for row order. A subscript is used to denote an element in a matrix and vector, for example x_{ij} and x_i. A vector $\mathbf{x}_j$ denotes the column j of matrix $\mathbf{X}$. A proposition is denoted as x while a numerical variable is denoted as x. A proposition x has two possible values *true* and *false* which is equivalent to a binary variable x which has values 1 and 0 respectively.

The transpose of a matrix or a vector is denoted by $\mathbf{X}^\top$ and $\mathbf{x}^\top$ respectively. In a group of numbers such as $1,\ 2, ..., , N$, $\backslash i$ denotes a subset of the group that contains all numbers except i. For different types of product, matrix/vector multiplication is denoted as $\mathbf{XY}$, while $\times$ denotes the scalar multiplication and $\circ$ denotes element-wise product. The notation = is used for assigning a value to a variable, while $\sim$ is used for sampling from a distribution.

Probability distribution of a variable x is denoted as $p(x)$. With a binary variable x, $P(x)$ denotes $p(x = 1)$ and $P(x|y)$ denotes $p(x = 1|y)$.

Logical connectives such as *NOT*, *AND*, *OR*, *XOR*, *IF-THEN*, and *IF-AND-ONLY-IF* are denoted using standard notation $\neg$,$\wedge$, $\vee$, $\oplus$, $\leftarrow$, $\leftrightarrow$ respectively. We use $\bigwedge_i \mathsf{x}_i$ to denote $\mathsf{x}_1 \wedge \mathsf{x}_2$.... and $\bigvee_i$ to denote $x_1 \vee x_2$.....

In a hierarchical structure, superscripts are used to denote the levels. For example, in a multilayer network a state of visible layer is denoted as $\mathbf{x}$ and state of a hidden layer l ($l > 0$) is denoted as $\mathbf{h}^{(l)}$. If a network has only two layers, $\mathbf{h}^{(1)}$ can be replaced by $\mathbf{h}$ for ease of presentation. The set of all parameters of a network is denoted as θ.

Abbreviations

aTPL	Adaptive Transferred Profile Likelihood
BM	Boltzmann Machine
CNF	Conjunctive Normal Form
CD	Contrastive Divergence
CIFF	Conjunctive If-And-Only-If
DBN	Deep Belief Network
DBM	Deep Boltzmann Machine
DNF	Disjunctive Normal Form
GSTL	Guided Self-taught Learning
KL	Kullback-Leibler
NN	Neural Network
NSCA	Neural-Symbolic Cognitive Agent
MI	Mutual Information
PCA	Principal Component Analysis
PLOFF	Penalty Logic Well-formed Formula
RBM	Restricted Boltzmann Machine
SAE	Stacked Auto-Encoder
SC	Sparse Coding
SDNF	Strict Disjunctive Normal Form
STL	Self-taught Learning
SVM	Support Vector Machine
WFF	Well-formed Formula

Chapter 1

Introduction

Unsupervised models such as restricted Boltzmann machines (RBMs) and deep belief networks (DBNs) can learn useful patterns for recognition tasks in wide range of domains. In addition, these patterns have been shown to capture domain representations at different levels. For example, visualisation of the patterns learned from handwritten image data indicates that low level patterns represent curves and edges while the higher level patterns represent more concrete shapes. This interesting characteristic of unsupervised learning intrigues a question of whether symbolic knowledge can also be represented by these patterns. This chapter takes this question as a starting point to propose a research on decomposition of representations in RBMs/DBNs to build a Learning, Extraction and Sharing (LES) system.

1.1 Motivation

RBMs, with many variations and extensions, are an efficient neural network model that has been applied very successfully recently as a building block for deep networks in diverse areas ranging from language generation to video analysis and speech recognition. Despite their success and the creation of increasingly complex network models and learning algorithms based on RBMs, the question of how knowledge is represented, and could be shared by such networks, has received comparatively little attention. Breiman argues that one can enjoy the effectiveness of complex systems while knowledge should be produced separately for explanation purposes [19]. This motivates the development of a learning, extraction and sharing system as illustrated in Figure 1 [19].

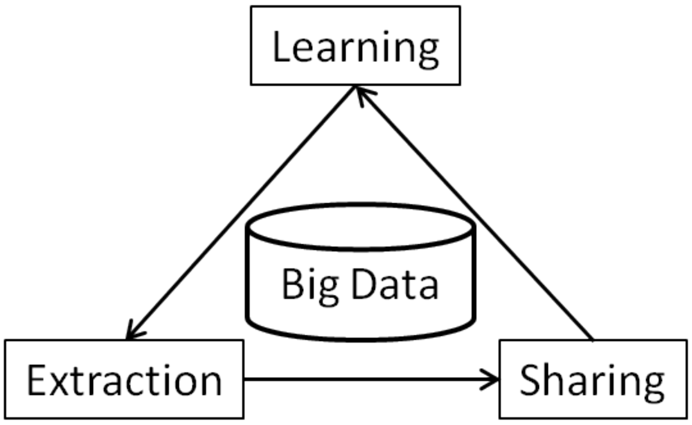
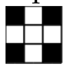


Figure 1.1: LES triangle: learning from data, knowledge extraction, and sharing for transfer learning.

The exponential growth of digital content has required the development of robust, flexible, modular and expressive systems in order to handle the challenges of Big Data. In order to achieve this while being able to provide interpretability, the ideal system should embody the capabilities of *learning*, *extraction* and *sharing* of knowledge given rich, complex data. More specifically, *learning* can automate the process of encoding knowledge hidden in a large data set, knowledge *extraction* from a trained model can further translate the knowledge into more readable forms such as symbolic or visual languages, and help highlight the relevant knowledge. It also promotes explicit reasoning, as will be exemplified in what follows. Finally, such knowledge can be used for *sharing*, i.e. to improve the learning in another related task. To realise all these capabilities one has to address the following challenging research questions:

- How should knowledge be represented?
- how can it be achieved from data?
- How to transfer it from one domain to another?

1.1.1 Knowledge Representation and Learning

Knowledge representation is traditionally concerned with "using formal symbols to represent a collection of propositions" [18]. Normally, the term "knowledge" refers to symbolic knowledge to which reasoning can be applied systematically. In classical logic, propositional logic is one of the most basic and popular symbolic languages, built upon propositions and the logical connectives “and”, “or”, “negation”, “implication”, and “bi-conditional”[118]. *Propositions* are particular kinds of sentences which only have either a *true* or *false* value. *Connectives* are the symbols which are used to construct complex sentences from simpler ones. For example, a 3×3 black and white image  may represent a “plus” sign. If the pixels are numbered in the order from left to right, top to bottom, the symbol x_i can be used to represent the proposition ”*pixel i is white*”. The symbol $\neg\mathsf{x}_i$ then represents the proposition “*pixel i is not white*”. Together with the proposition that there are only white or black pixels in this case, the above propositions can be reasoned with to conclude that “*pixel i is black*”. Let the symbol plus denote the proposition “*the picture is a plus sign*”. In a closed world with 9 variables denoting the values of the 9 pixels, knowledge about the plus sign can be represented by the following logical rule, indicating that plus is true if x_1 is false, x_2 is true, etc:

$$\mathsf{plus} \leftarrow \neg\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg\mathsf{x}_3 \wedge \mathsf{x}_4 \wedge \mathsf{x}_5 \wedge \mathsf{x}_6 \wedge \neg\mathsf{x}_7 \wedge \mathsf{x}_8 \wedge \neg\mathsf{x}_9$$

In other words, we have used a set of symbols to represent visual information. By encoding knowledge into a symbolic form, one may be able to reason about such knowledge in a precise way which follows the rules of logical inference [117].

In order to support more flexible reasoning, however, especially under uncertainty, knowledge can be represented by probabilistic models. For example, a Bayesian network [103] encodes knowledge into a dependency graph and probability tables. Let us consider the same “plus sign” example above, where we can treat a pixel i as a binary variable $x_i \in \{0, 1\}$, with $x_i = 1$ indicating that the pixel is white and $x_i = 0$ indicating that it is black; $plus \in \{0, 1\}$ indicates whether the picture is a plus sign ($plus = 1$) or not ($plus = 0$). A Bayesian network represents the knowledge in this domain as a graphical model with distribution table as shown in Table 1.1. The table assigns a high probability of 0.8 to the “plus” shape, a probability of 0.022 to all the shapes that have only one pixel different from the ”plus” shape, and probability 0 to all the other shapes.

In a complex domain with high dimensional, real-valued data, the knowledge

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	$p(plus = 1 \mid x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$
0	0	0	0	0	0	0	0	0	0
...									
0	1	0	1	1	1	0	0	0	0.022
0	1	0	1	1	1	0	0	1	0
0	1	0	1	1	1	0	1	0	0.8
0	1	0	1	1	1	0	1	1	0.022
...									
1	1	1	1	1	1	1	1	1	0

Table 1.1: Probability table of the "plus" shape graphical model.

representation is more complex. For example, the probability that the picture is a plus sign given the values of a large number of pixels is

$$P(plus|\mathbf{x}) = \mathbf{pdf}(\mathbf{x}, \theta)$$

where **x** is a vector denoting the set of all variables and **pdf** is a probability distribution function.

Learning algorithms applied to neural networks [49] and Bayesian networks [103] have been shown capable of representing rich knowledge, being particularly useful in the case of noisy data. Neural networks, however, in spite of their success, are difficult to interpret. The area of knowledge extraction [6, 136, 31] seeks to address this problem mainly by translating the networks into symbolic knowledge. Knowledge extraction has been normally applied to feed-forward neural networks trained in supervised fashion using the back-propagation learning algorithm [115, 116, 75].

More recently, research has shown that the use of unsupervised models [100, 52, 54, 78, 77, 111, 11], may improve the performance of network models at learning structures from complex data, whereby patterns of interest are captured by basis vectors. This area became known as representation learning, due to its original goal of being able to stack unsupervised models on top of each other in order to learn progressively more complex levels of representations directly from data [54, 11]. Normally, such deep network models encode knowledge in the form of weight matrices. In this thesis, we investigate symbolic knowledge extraction from such unsupervised network models. We decompose the weight matrices into a set of symbolic knowledge rules called *confidence rules*, which are associated with a real number, as done by Pinkas in the case of Penalty Logic rule extraction for Hopfield networks [109]. The symbolic form is expected to provide insight into the representation and reasoning taking place within stacks of restricted Boltzmann machines, while the associated real values account for reasoning under uncertainty.

A *confidence rule* is an if-and-only-if statement associated with a real number c, written $c : \mathsf{h} \leftrightarrow \mathsf{b}$, where h is called a hypothesis and b is a conjunction of propositions. For example, uncertain knowledge about the plus sign example can be represented symbolically as:

$$0.8 : \mathsf{h} \leftrightarrow \neg \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg \mathsf{x}_3 \wedge \mathsf{x}_4 \wedge \mathsf{x}_5 \wedge \mathsf{x}_6 \wedge \neg \mathsf{x}_7 \wedge \mathsf{x}_8 \wedge \neg \mathsf{x}_9 \wedge \mathsf{plus}$$

This rule can be read as *"If* p_1 *is false,* p_2 *is true,* p_3 *is false,* p_4 *is true,* p_5 *is true,* p_6 *is true,* p_7 *is false,* p_8 *is true and* p_9 *is false, assuming that the hypothesis* h *holds, then* plus *should be true with confidence* 0.8".

Inference using confidence rules, as will be defined precisely in **Chapter 4**, is done by finding the truth-value of plus that maximises the sum of the confidence values of the rules that are satisfied with $\mathsf{h} = \mathit{true}$.

In this thesis, confidence rules will be extracted from stacks of RBMs in a modular way, and the inference rule referred to above will be used to allow symbolic hierarchical reasoning and representation under uncertainty. The results of two approaches, namely partial models and complete models, both defined in this thesis, will be evaluated on image domains.

Partial-models offer a compact representation for RBMs, a set confidence rules, which is to be used for hierarchical inference, as defined in **Chapter 4**. For example:

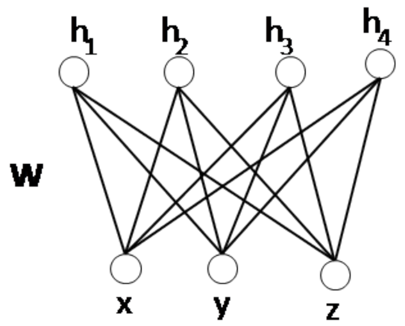
$$c^{(1)} : \mathsf{h} \leftrightarrow \neg \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg \mathsf{x}_3 \wedge \mathsf{x}_4 \wedge \mathsf{x}_5 \wedge \mathsf{x}_6 \wedge \neg \mathsf{x}_7 \wedge \mathsf{x}_8 \wedge \neg \mathsf{x}_9$$
$$c^{(2)} : \mathsf{plus} \leftrightarrow \mathsf{h}$$

Complete-models are a richer form of representation, resembling the set of weights in an RBM closely, where each proposition in b can also be associated with a real number, for example $(\beta : \mathsf{b})$. Intuitively, this real number can be seen as indicating the relative importance of the proposition. An example of a confidence rule in a complete model would be:

$$\begin{aligned} c : \mathrm{h} \leftrightarrow\ & (\beta_1 : \neg \mathrm{x}_1) \wedge (\beta_2 : \mathrm{x}_2) \wedge (\beta_3 : \neg \mathrm{x}_3) \wedge (\beta_4 : \mathrm{x}_4) \\ & \wedge (\beta_5 : \mathrm{x}_5) \wedge (\beta_6 : \mathrm{x}_6) \wedge (\neg\beta_7 : \mathrm{x}_7) \wedge (\beta_8 : \mathrm{x}_8) \wedge (\beta_9 : \neg \mathrm{x}_9) \end{aligned}$$

Both *partial-models* and *complete-models* can be obtained from a weight matrix by converting each column vector into a confidence rule. However, *partial-models* are more "symbolic" by having fewer associated real values than *complete-models*, which in turn should capture better the influence of the observed variables onto the hidden variables of an RBM.

Example 1.1.1. Suppose a restricted Boltzmann machine with three visible units (x, y, z) and four hidden units (h_1, h_2, h_3, h_4), as shown in the figure below, has been trained from data, leading to a set of trained parameters as shown in the weight matrix **W**.



The weight matrix of this RBM is: $\mathbf{W} = \begin{pmatrix} -6.5591 & -5.7882 & 0.8857 & 1.5601 \\ -6.6418 & 0.7022 & -6.4277 & 1.386 \\ -6.5909 & 0.7011 & 0.7538 & -0.39 \end{pmatrix}$

The probability of a state of the visible layer being true is inversely proportional to a free energy function[53], as $p(x, y, z) \propto \exp(-\mathcal{F}(x, y, z))$ with:

$$\mathcal{F}(x, y, z)) = -\sum_{j=1}^{4} \log(1 + \exp(w_{1j}x + w_{2j}y + w_{3j}z))$$

From this model, one can extract confidence rules efficiently, directly from the weight matrix (the extraction algorithms will be introduced and investigated in detail in **Chapter 3**, **Chapter 4** and **Chapter 5**). In this example, let us take the simple case where each rule is extracted from a column vector of the weight matrix, such that positive weights are converted into a positive proposition x, and negative weights into a negative proposition $\neg\mathsf{x}$. A confidence value for the rule is then computed as the average of the absolute values of the weights in the column vector. For example, a vector $[-6.5591\ -6.6418\ -6.5909]$ will be converted into $6.5972 : \mathsf{h}_1 \leftrightarrow \neg\mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z}$. As a result, the set of confidence rules extracted from the weight matrix **W** is:

$$R_{confidence} = \begin{array}{l} 6.5972 : \mathsf{h}_1 \leftrightarrow \neg\mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z} \\ 2.3972 : \mathsf{h}_2 \leftrightarrow \neg\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z} \\ 2.6891 : \mathsf{h}_3 \leftrightarrow \mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z} \\ 1.1120 : \mathsf{h}_4 \leftrightarrow \mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z} \end{array} \tag{1.1}$$

where h_j is a hypothesis, and $\mathsf{x}, \neg\mathsf{x}, \mathsf{y}, \neg\mathsf{y}, \mathsf{z}, \neg\mathsf{z}$ are propositions indicating that $x = 1, x = 0, y = 1, y = 0, z = 1$ and $z = 0$, respectively.

With the confidence rules, one can, for example, apply weighted MAX-SAT [112, 50] to decide on the propositions which will give the highest satisfiability of the hypotheses being *true*. Similar reasoning can also be done in the RBM. Given the states of two of the inputs, the state of the third input will seek to maximise the joint probability $p(x, y, z)$. For example, given $x = 1$, $y = 0$ then $z = 1$ because $\mathcal{F}(x = 1, y = 0, z = 1) = -3.263 < \mathcal{F}(x = 1, y = 0, z = 0) = -2.986$, implying that $p(x = 1, y = 0, z = 1) > p(x = 1, y = 0, z = 0)$.

If we consider z to be a target variable, i.e. a label unit, we can separate the weight matrix into a lower-level weight matrix (between visible units (x,y) and the hidden layer) and a higher-level weight matrix (between the hidden layer and the target z). Applying the same extraction of confidence rules to these matrices, one obtains two sets of rules to which hierarchical reasoning can be applied, as discussed in more detail in **Chapter 4**, as follows:

$$R^{(low)}_{partial} = \begin{array}{l} 6.6004 : \mathsf{h}_1 \leftrightarrow \neg\mathsf{x} \wedge \neg\mathsf{y} \\ 3.2452 : \mathsf{h}_2 \leftrightarrow \neg\mathsf{x} \wedge \mathsf{y} \\ 3.6567 : \mathsf{h}_3 \leftrightarrow \mathsf{x} \wedge \neg\mathsf{y} \\ 1.4730 : \mathsf{h}_4 \leftrightarrow \mathsf{x} \wedge \mathsf{y} \end{array}$$

$$R^{(high)}_{partial} = \begin{array}{l} 6.5909 : \mathsf{z} \leftrightarrow \neg\mathsf{h}_1 \\ 0.7011 : \mathsf{z} \leftrightarrow \mathsf{h}_2 \\ 0.7538 : \mathsf{z} \leftrightarrow \mathsf{h}_3 \\ 0.3900 : \mathsf{z} \leftrightarrow \neg\mathsf{h}_4 \end{array}$$

Now, by inspecting the symbolic part of the rules, one finds that, for example, $\neg\mathsf{x} \wedge \neg\mathsf{y} \leftrightarrow \mathsf{h}_1$ and $\mathsf{z} \leftrightarrow \neg\mathsf{h}_1$ is equivalent to $\neg\mathsf{x} \wedge \neg\mathsf{y} \leftrightarrow \neg\mathsf{z}$. This rule is more discriminative in that it represents a relationship between a group of non-target variables and a target variable.

By repeating the above process, the symbolic form of the rules extracted from the RBM are: $\neg\mathsf{x} \wedge \neg\mathsf{y} \leftrightarrow \neg\mathsf{z}$, $\neg\mathsf{x} \wedge \mathsf{y} \leftrightarrow \mathsf{z}$, $\mathsf{x} \wedge \neg\mathsf{y} \leftrightarrow \mathsf{z}$, $\mathsf{x} \wedge \mathsf{y} \leftrightarrow \neg\mathsf{z}$ which represent the XOR function $x \oplus y \leftrightarrow z$. The dataset on which the above RBM was trained was indeed obtained from this XOR function.

Our hypothesis is that a hierarchical representation and reasoning algorithms can provide insight into the relevance of the knowledge learned by stacks of RBMs and facilitate transfer learning, as a result.

Even though the combination of rules, as used above, will be shown in some cases not to be effective for improving prediction accuracy (particularly in complex image domains), the rules will be shown to offer a compact representation that is useful for transfer learning, i.e. to improve performance in a related or analogous domains through a better network initialisation.

Let us now consider the complete-models which would be obtained from the weight matrix **W**:

$$
R^{(low)}_{complete} = \begin{array}{l}
6.6004 : \mathsf{h}_1 \leftrightarrow (0.9937 : \neg\mathsf{x}) \wedge (1.0063 : \neg\mathsf{y}) \\
3.2452 : \mathsf{h}_2 \leftrightarrow (1.7827 : \neg\mathsf{x}) \wedge (0.2164 : \mathsf{y}) \\
3.6567 : \mathsf{h}_3 \leftrightarrow (0.2422 : \mathsf{x}) \wedge (1.7578 : \neg\mathsf{y}) \\
1.4730 : \mathsf{h}_4 \leftrightarrow (1.0591 : \mathsf{x}) \wedge (0.9409 : \mathsf{y})
\end{array}
$$

$$
R^{(high)}_{complete} = \begin{array}{l}
6.3909 : \mathsf{z} \leftrightarrow (1 : \neg\mathsf{h}_1) \\
0.7011 : \mathsf{z} \leftrightarrow (1 : \mathsf{h}_2) \\
0.7538 : \mathsf{z} \leftrightarrow (1 : \mathsf{h}_3) \\
0.3900 : \mathsf{z} \leftrightarrow (1 : \neg\mathsf{h}_4)
\end{array}
$$

One can see that each confidence rule in a complete-model accounts for a basis vector (a column vector of the weight matrix). For example, vector [-6.5591 -6.6418] is converted to rule $6.6004 : \mathsf{h}_1 \leftrightarrow (0.9937 : \neg\mathsf{x}) \wedge (1.0063 : \neg\mathsf{y})$, where the rules confidence is $(6.5591 + 6.6418)/2 = 6.6004$, and the associated values of the two propositions $\neg\mathsf{x}$ and $\neg\mathsf{y}$ are $6.5591/6.6004 = 0.9937$ and $6.6418/6.6004 = 1.0063$, respectively. Notice that the confidence values c of the confidence rules in a complete-model are the same as in the partial-model of an RBM. With the confidence values of the propositions accounted for, the rules in a complete model capture exactly the column vectors of **W**. Therefore, in practice we only need to compute the confidence values of the rules and we can use the column vectors of the weight matrix as representation of *complete-models*.

1.1.2 Knowledge Sharing and Transfer Learning

Following knowledge extraction, confidence rules can be used for reasoning, e.g. if x is true and y is false, given conflicting rules $0.5 : \mathsf{h} \leftrightarrow \mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z}$ and $0.01 : \mathsf{h} \leftrightarrow \mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z}$, one can be more confident that z is true than it is false. Confidence rules can also be used to support learning in a related domain. In AI, knowledge sharing has emerged recently as an important research topic [137, 31, 101, 135, 62]. In neural networks, the idea of encoding knowledge is not new; symbolic knowledge can be encoded into the initial set of weights of a neural network in order, hopefully, to improve on an otherwise random network initialisation [137, 31]. One might see this as a continuous process of using reliable knowledge provided by experts to guide the learning of new knowledge from data, as more and more data becomes available.

To enable the above knowledge refinement or, more generally, knowledge sharing across different (but related) domains, a system should require: source domain(s) for which knowledge is provided; target domain(s) that reuse the knowledge; and a transfer mechanism. For example, suppose that knowledge about a minus sign is to be used to learn new knowledge about the plus sign (as seen in an earlier example). A transfer mechanism needs to be designed with useful mapping rules. For example, a rule for a minus sign might be[1]:

$$\mathsf{minus} \leftrightarrow \mathsf{x}_4 \wedge \mathsf{x}_5 \wedge \mathsf{x}_6$$

This rule can be used when learning new knowledge about the plus sign, as follows:

$$\mathsf{plus} \leftrightarrow \mathsf{minus} \wedge \mathsf{x}_2 \wedge \mathsf{x}_8$$

The transfer mechanism will depend on the transfer medium, i.e. the language/model in which knowledge is represented. It does not require the use of logic rules in every case, and the mapping rules can be created manually by experts; however, this is a daunting task. When a mechanism automatically learns the mapping rules then this is called *transfer learning* [101, 135, 62]. In this thesis, knowledge extraction and the proposed confidence rules language will be shown useful as part of a new transfer learning algorithm, which will be shown empirically to be an effective medium for transfer learning.

1.2 Objectives

The objectives of this research are to propose, develop and evaluate a new form of knowledge representation for stacks of RBMs and to apply it to knowledge

[1]This rule uses negation by default [31]: if a proposition x_i does not appear in a rule then its negation $\neg\mathsf{x}_i$ is assumed to be true by default, unless stated otherwise

extraction, neural-symbolic integration and transfer learning.

> **Hypothesis Statement** *The decomposition of complex RBM representations into logic-based propositions provides an effective way of achieving knowledge extraction, insertion and transfer between RBMs.*

Specifically, this research addresses the following questions: (i) *how knowledge learned by an RBM can be represented symbolically?*; (ii) *how learning from data and symbolic knowledge can be integrated into a neural-symbolic system to improve performance?*; and (iii) *how symbolic knowledge can be used to improve learning from data in a different, related domain?*.

We show that confidence rules offer an adequate hierarchical decomposition for the set of weight matrices of deep belief networks (DBNs). We introduce two types of rules: *partial-models* and *complete-models* as briefly discussed in §1.1. The idea behind *partial-models* is to offer a compact language for knowledge extraction and insertion into DBNs. With *complete-models*, a symbolic rule captures exactly the information in a basis vector. Such representation can be adapted to improve state-of-the-art results in transfer learning, by adapting, according to the confidence values, symbolic knowledge from a domain to data from another.

1.3 Contributions

The main contribution of this thesis is the development and evaluation of a prototype of the Learning, Extraction and Sharing system based on deep belief networks, as described above.

A new form of knowledge representation for stacks of RBMs, called "confidence rules", is introduced. It combines logical reasoning and quantitative reasoning. It is shown to be an adequate and compact representation for a type of hierarchical probabilistic connectionist system, namely RBMs. Confidence rules are the core component in the proposed Learning, Extraction and Sharing system. They support knowledge extraction from deep belief networks, and enable knowledge evaluation and sharing.

A new efficient algorithm for extracting symbolic knowledge in the form of confidence rules from DBNs is introduced and evaluated. The key idea of this extraction algorithm is to decompose the weight matrix in each layer of the DBN into a set of independent symbolic elements. The extraction is shown to be useful by offering a compact representation, modular organisation of the rules, and support for hierarchical inference in the presence of uncertainty. This type of inference is more flexible than classical logical inference through the use of real-valued confidence values. It is shown experimentally in image domains that confidence rules can save significant amounts of memory in comparison to RBMs while still guaranteeing performance at feature extraction tasks.

A new neural-symbolic system integrating symbolic knowledge and DBNs is proposed and evaluated. By using confidence rules as an intermediate language, we translate symbolic rules, serving as background knowledge, into a hierarchical set of weight matrices. During network training, we employ the rule inference to guide the learning. The idea of encoding symbolic knowledge into a connectionist system to improve learning is not new. However, this is to the best of our knowledge, the first neural-symbolic system for unsupervised learning and modular reasoning using RBMs. We show on experiments using DNA sequence analysis and the MNIST handwritten digits datasets that the encoding of knowledge can help improve learning performance and inference in deep belief networks.

We propose and evaluate a method for using confidence values for representation ranking. The method computes a confidence value as the mean of the absolute values of the basis vectors corresponding to a representation. We measure the usefulness of confidence values using: visualisations of the reconstructed images, classification accuracy, and mutual information. The results show that the representations with the highest confidence values capture the majority of an RBM.

Finally, we propose a new transfer learning algorithm based on the idea of using prior knowledge to guide the learning in a deep neural-symbolic system and the above representation ranking. Instead of using background knowledge in the same domain, we develop an algorithm to reuse representations with high confidence values from a source domain in a target domain. This is possible because high-ranking representations are chosen for transferring and such representations can be adapted as part of the learning process at the target domain, based on the data available at the target domain. We test the algorithm by transferring representations from source RBMs trained on handwritten letters onto target RBMs trained to recognise handwritten digits. The proposed transfer algorithm is shown to outperform state-of-the-art self-taught learning and combinations of self-taught learning and RBM learning. Each confidence rule is associated with an adaptation factor which becomes a parameter for training in the target domain, allowing representations to be transformed progressively. Furthermore, the use of confidence rules offers an approach to deal with the problem of "biased sampling". The "biased sampling" problem happens when many representations to be transferred make the learning in the target domain dependent on the source domain. By transferring only a small number of representations with high confidence values, using confidence values and mutual information, or applying dropout onto a small set of transferred representations for each batch learning in the target domain, biased sampling can be reduced. Extensive experiments confirm that the use of knowledge and adaptation factors can improve the effectiveness of the transferred representations with the target domains. Our experiments also show that transfer learning is more effective when knowledge, and not data, is transferred.

Publications

1. Son N. Tran and Artur d'Avila Garcez. Deep Belief Logic Networks. (submitted)
2. Son N. Tran and Artur d'Avila Garcez. Adaptive Transferred-profile Likelihood Learning. (submitted)
3. Hazrat Ali, Son N. Tran, Emmanouil Benetos, Xianwei Zhou. Hybrid Representation Learning for Speaker Recognition. (submitted)
4. Son N. Tran, Srikanth Cherla, Artur d'Avila Garcez, Tillman Weyde. Probabilistic Approach for Relative Similarity. (in preparation)
5. Son N. Tran, Artur S. d'Avila Garcez. Efficient Representation Ranking for Transfer Learning. *In International Joint Conference on Neural Network*. Killarney, Ireland, 2015.
6. Hazrat Ali, Son N. Tran, Artur S. d'Avila Garcez, Tillman Weyde. Convolutional Data: Towards Deep Audio Learning from Big Data. *In 1st UCL Workshop on the Theory of Big Data*. London, UK. 2015.
7. Son N. Tran and Artur d'Avila Garcez. Low-cost Representation for Restricted Boltzmann Machine. *In 21st International Conference on Neural Information Processing*. Kunching, Malaysia, 2014.

8. Thanh Vu, Dawei Song, Alistair Willis, Son N. Tran. Improving Search Personalisation with Dynamic Group Formation. *In SiGIR*. Australia, 2014.
9. Son N. Tran, Emmanouil Benetos and Artur d'Avila Garcez. Learning Motion-Difference Features using Gaussian Restricted Boltzmann Machines for Efficient Human Action Recognition. *In International Joint Conference on Neural Network*. Beijing, China, 2014.
10. Son N. Tran, Daniel Wolff, Tillman Weyde, and Artur d'Avila Garcez. Feature Preprocessing with Restricted Boltzmann Machine for Music Similarity Learning. *In Audio Engineering Society 53rd conference on Semantic Audio*. London, UK, 2014. (**Winner of Reproducible Prizes**).
11. Hazrat Ali, Artur d'Avila Garcez, Son N. Tran, Xianwei Zhou. Hybrid Features Combination for Audio Data Classification. *In Machine Learning and Data Analytics Symposium*, 3-4 March, Doha, Qatar, 2014.
12. Son N. Tran and Artur d'Avila Garcez. Knowledge Extraction from Deep Belief Networks for Images. *In IJCAI-2013 Workshop on Neural-Symbolic Learning and Reasoning*. Beijing, China, 2013.
13. Son Tran and Artur Garcez. Logic Extraction from Deep Belief Networks. *In ICML2012 Representation Learning Workshop*. Edinburgh,UK, July 2012

Software and code

Link: `https://github.com/sFunzi/`

1. RepDeepLearn: Implementation of representation/deep learning and reasoning models such as restricted Boltzmann machines (RBMs), Auto Encoders, Non-negative Matrix Factorization, Sparsity, deep belief networks (DBNs), deep Boltzmann machines (DBMs), Neural Networks
2. ConfidenceLogic: Knowledge Extraction from RBMs and DBNs
3. Motion-Difference: Action recognition
4. RelSim: Relative similarity models, tested on music data
5. ATPL: Extraction and transfer of representation from learned RBMs

1.4 Organisation of the Thesis

The first chapter (this chapter) introduces the concept of knowledge learning-extraction-sharing, the features of the system, objectives, research questions and contributions of the work.

Chapter 2 introduces the deep models used for *LES: Learning-Extraction-Sharing*. We review the theory of representation learning, RBMs and deep learning using unsupervised models. We illustrate the use of stacks of RBMs in a range of applications in music similarity, action recognition, speaker recognition and melody modelling. We review the related work on knowledge-based neural networks, knowledge extraction and transfer learning.

Chapter 3 reviews the related work on Penalty Logic and extends Pinkas results to DBNs by showing that propositional calculus is equivalent to minimising a DBNs energy function. We observe that the signs of the weights already represent the logical propositions, which will serve as the basis for an efficient extraction algorithm. We introduce the concept of confidence rules formally and show that confidence rules can be approximated by training a DBN. This indicates that the extraction of confidence rules from DBNs is promising, which will then be investigated empirically.

In **Chapter 4**, we introduce the concepts of *partial-models* and *complete-models* formally. We present an algorithm for the extraction of *partial-models* from RBMs and DBNs. We then empirically investigate the effectiveness of the extracted *partial-models* in terms of representation and inference. Based on the results, we also propose an encoding algorithm to integrate symbolic background knowledge and learning in DBNs.

In **Chapter 5** we investigate the use of confidence values for representation ranking and transfer learning. The representations are seen as a set of *complete-models* extracted from a domain to be ranked and transferred to improve learning in another domain. We show that confidence values can be used to rank the representations which are then to be transferred.

In **Chapter 6** we tackle the problem of knowledge reuse and propose a general framework for knowledge reuse and transfer. The framework is based on the idea of profile likelihood with an assumption that part of the parameters are transferred and adapted from another domain. Each *complete-model* is associated with an adaptive factor which is responsible for transforming the *complete-model* onto the target domain. Experimental results are analysed extensively, indicating that the proposed transfer method can improve on the state-of-the-art.

Chapter 7 concludes the thesis, summarises the contributions and discusses directions for future work.

Chapter 2

Background

Recent research has shown that an emerging technique called deep learning can be effective in vision, audio, and text domains. Furthermore, with an effective layer-wise unsupervised learning, a deep network can learn a hierarchy of concepts from data. This chapter reviews deep networks - its building block restricted Boltzmann machines and deep belief networks - and applications using unsupervised learning. It also reviews related work on knowledge-based neural networks and transfer learning.

2.1 The Importance of Unsupervised Learning in Deep Learning

Deep beliefs networks are one among many machine learning models which have been categorised as "deep networks" [64, 75, 54, 11, 12, 119, 120, 122, 143]. The term "deep network" usually refers to a connectionist system which has many hidden layers. An early deep network model was the multi-layer artificial neural network (ANN) [64, 65]. However, training deep ANNs is not easy due to a problem called "vanishing/exploding gradient" with the back-propagation algorithm [59]. This problem can be alleviated through unsupervised layer-wise learning [122].

With more attention paid to deep learning and further study of unsupervised layer-wise learning [38, 56, 132], recent research indicates that it is possible to train a deep network purely by supervised algorithms using rectified linear units [93, 45]. While supervised deep learning is shown to be adequate to the effective learning of input-output mappings given large amounts of data, some researchers remain concerned about the role of unsupervised learning for the following reasons.

First, it has been shown through theoretical and experimental results that supervised learning is not always preferred over unsupervised learning [97]. In particular, even though unsupervised learning tends to achieve higher error (lower accuracy) it can converge faster than supervised learning in many cases. In deep learning, a well known experiment in the Google Brain project showed that it is possible to train a classifier without providing any label by stacking shallow models one on top of another [73].

Second, Bottou has raised a question about "a new path to AI" in that we can "algebraically enrich the set of manipulations applicable to training systems, and build reasoning capabilities from the ground up" [17]. Especially, in the Nature Review paper [143], Lecun, Bengio and Hinton have expressed their expectation that unsupervised learning in deep networks should become more important. This has been echoed by most of the panellists in the Panel Discussion at ICML 2015 Deep Learning workshop [1].

In this thesis we focus on DBNs, unsupervised models of deep learning created by stacking restricted Boltzmann machines on top of each other [54], as specified below.

2.2 Energy-based Connectionist Systems

Connectionist systems normally refer to a set of models made by interconnected networks of neurons (or units) [124, 51]. An energy based connectionist system (ECS) N is a neural network with bidirectional connections which is characterised by an energy function:

$$E_N(\mathbf{x}) = -\sum_{\forall i \forall j > i} f_{ij}(\mathbf{x}) - \sum_{\forall i} g_i(\mathbf{x}) \tag{2.1}$$

[1] http://deeplearning.net/2015/07/13/a-brief-summary-of-the-panel-discussion-at-dl-workshop-icml-2015/

where f_{ij} and g_i are potential functions for the state $\mathbf{x}$ of the model. There exist different types of ECSs depending on how this function is defined. This also characterises a probability distribution of a model as:

$$p(\mathbf{x}) = \frac{e^{-E(\mathbf{x})/T}}{Z} \tag{2.2}$$

where T is the temperature and $Z = \sum_{\mathbf{x}} e^{-E(\mathbf{x})/T}$ is a partition function.

The probability of a unit i being activated ($x_i = 1$) given the states of some other units $\mathbf{x}_{j\subset\setminus i}$ is:

$$P(x_i|\mathbf{x}_{j\subset\setminus i}) = \sum_{\mathbf{x}_{k\neq j}} p(x_i = 1, \mathbf{x}_{k\neq j}|\mathbf{x}_{j\subset\setminus i}) \tag{2.3}$$

where $\mathbf{x}_{j\subset\setminus i}$ denotes a subset of the units which does not contain x_i, $\mathbf{x}_{\setminus i}$ denotes all the units in the model except x_i, and $\mathbf{x}_{k\neq j}$ is another subset such that $\mathbf{x}_{\setminus i} = \mathbf{x}_{j\subset\setminus i} \cup \mathbf{x}_{k\neq j}$ and $\varnothing = \mathbf{x}_{j\subset\setminus i} \cap \mathbf{x}_{k\neq j}$.

In what follows, we recall three well-known instances of the above model: Hopfield networks, Boltzmann machines and restricted Boltzmann machines, all belonging to the same family of potential functions $f_{ij}(\mathbf{x}) = w_{ij}x_ix_j$ and $g_i(\mathbf{x}) = s_ix_i$, where w_{ij} is the connection weight between units x_i and x_j, and s_i is a bias for x_i. If function f consists of the product of more than two units, e.g $f_{ijk}(\mathbf{x}) = w_{ijk}x_ix_jx_k$, then the model is called "higher-order". These models can be seen as generative models [83, 97, 10] which represent a joint probability between the variables and which can be trained by unsupervised algorithms. The term "generative" is used in this context to distinguish from "discriminative" models which represent a conditional probability of the data given a label variable [97, 70]. Discriminative models are usually trained by supervised learning algorithms.

The connection weights in a Hopfield network, Boltzmann machine, or restricted Boltzmann machine are said to be symmetrical, i.e. having symmetric connections such that the weight from unit i to unit j is the same as the weight from unit j to unit i. Such network models are for this reason called symmetrical networks [40, 82].

2.2.1 Hopfield Networks

A Hopfield network [61] is a neural network with recurrent connections. The state of each unit (or neuron) can be 0 or 1, where state 0 indicates "not firing" and state 1 indicates "firing". One may see a Hopfield network as an energy-based connectionist system, with temperature $T = 0$ [57], and therefore the inference rule in Eq. 2.3 becomes deterministic.

$$x_i = \begin{cases} 1 & \text{if } \sum_j w_{ij}x_j + s_i > 0; \\ 0 & \text{otherwise }; \end{cases} \tag{2.4}$$

Starting from an initial state $\mathbf{x}^{(0)}$, the model can iteratively update to a final state that minimises the function in Eq. 2.1. It can also be seen as a Markov chain of a symmetric connectionist system with zero temperature. This property makes Hopfield networks able to act as associative memory systems where each memory state is a local minimum of the energy function. An example of Hopfield network is shown in Figure 2.1.

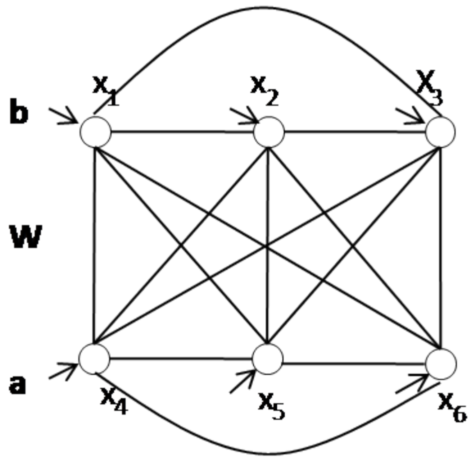


Figure 2.1: A Hopfield network ($\mathbf{s} = \{\mathbf{a}, \mathbf{b}\}$).

2.2.2 Boltzmann machines

Boltzmann machines (BMs) [57] are energy-based connectionist systems in which the temperature $T \neq 0$. We use I and J to denote the number of observed (visible) units and unobserved (hidden) units respectively. As in the case of the Hopfield network, the weight matrix of a Boltzmann machine is symmetric, but it can be expressed in terms of $w_{ij}^{(xh)}, w_{ii'}^{(xx)}, w_{jj'}^{(hh)}$ to denote, respectively, the connection weight between unit x in the visible layer and unit h in the hidden layer, the weight between two units in the visible layer, and the weight between two units in the hidden layer, as shown in Figure 2.2. We also use a_i and b_j to denote the biases of visible unit i and hidden unit j. The energy function of a Boltzmann machine then becomes:

$$E_{BM}(\mathbf{x}, \mathbf{h}) = -\sum_{i,j}^{I,J} w_{ij}^{(xh)} x_i h_j - \sum_{i,i'>i}^{I,I} w_{ii'}^{(xx)} x_i x_{i'} - \sum_{j,j'>j}^{J,J} w_{jj'}^{(hh)} h_j h_{j'} - \sum_{i}^{I} a_i x_i - \sum_{j}^{J} b_j h_j \quad (2.5)$$

A Boltzmann machine can be trained by maximising the log-likelihood:

$$\text{Minimise } \mathcal{L}_N = \log p(\mathcal{D}) \quad (2.6)$$

where $\mathcal{D}$ is the observed data from a domain. Inference of the state of a unit in one layer given the state of the other layer is intractable if the number of units in this layer is large. Normally, then, one can use Gibbs sampling [23] over the network to get data samples from the marginal distribution $p(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h})$ until equilibrium is reached.

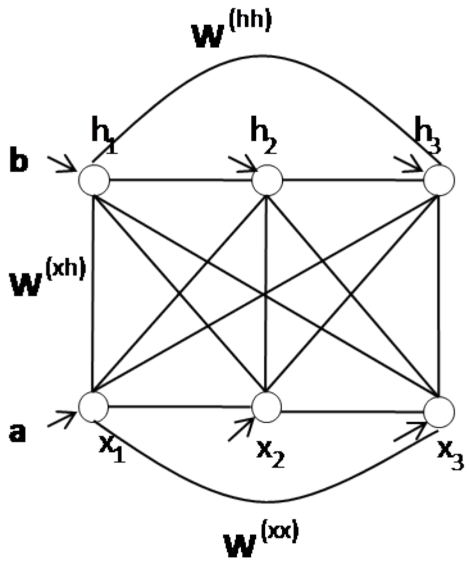


Figure 2.2: A Boltzmann machine.

Learning this type of model is also difficult because the log-likelihood exact calculation is intractable. Traditional approaches deal with this problem by using Markov Chain Monte Carlo (MCMC) methods to sample the states of units, which are needed for approximating the gradients [94, 44]. However, MCMC methods are usually computationally expensive and their convergence time is hard to predict. In order to reduce the inference time in the training, one may use variational methods [67] to approximate the states of the units.

The gradient of the log-likelihood function in Eq. 2.6 is:

$$\nabla\theta = \mathbb{E}\Big[\frac{\partial E_{BM}(\mathbf{x},\mathbf{h})}{\partial\theta}\Big]_{\mathbf{h}|\mathbf{x}} - \mathbb{E}\Big[\frac{\partial E_{BM}(\mathbf{x},\mathbf{h})}{\partial\theta}\Big]_{\mathbf{x},\mathbf{h}} \tag{2.7}$$

The first term gives an expectation of the gradient over the conditional distribution $p(\mathbf{h}|\mathbf{x})$, and the second term, the expectation over the joint distribution $p(\mathbf{x},\mathbf{h})$. In Boltzmann machines, these expectations are both intractable. To learn the model, a more recent and popular alternative to variational methods, is the Contrastive Divergence (CD) algorithm [52]. CD is an efficient algorithm, to approximate good parameters for the model. The CD algorithm approximates the negative log-likelihood by minimising the difference of the two Kullback-Leibler divergences [69, 52].

$$\mathrm{KL}(p(\mathbf{x}_{\mathcal{D}})\|p(\mathbf{x};\theta)_k) - \mathrm{KL}(p(\mathbf{x};\theta)_k\|p(\mathbf{x};\theta)_\infty) \tag{2.8}$$

where $p(\mathbf{x}_{\mathcal{D}})$ is the data distribution, $p(\mathbf{x};\theta)_k$ and $p(\mathbf{x};\theta)_\infty$ are the model distribution after a k step Gibbs sampling and the model distribution at equilibrium state, respectively.

2.2.3 Restricted Boltzmann Machines

A restricted Boltzmann machine (RBM) with I visible units and J hidden units has energy function:

$$E_{RBM}(\mathbf{x},\mathbf{h}) = -\sum_{i,j}^{I,J} w_{ij}x_ih_j - \sum_{i}^{I} a_ix_i - \sum_{j}^{J} b_jh_j \tag{2.9}$$

RBMs are a simplified version of the Boltzmann machine for which there are no connections between units in the same layer, as shown in Figure 2.3. With this

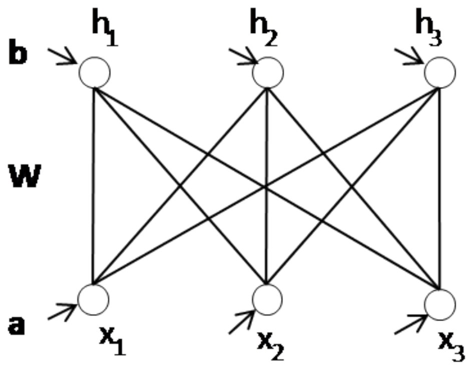


Figure 2.3: A restricted Boltzmann machine.

constraint, the calculation of the probability of a unit in a layer being activated (i.e. =1) given the state of the other layer becomes tractable :

$$
\begin{aligned}
P(x_i|\mathbf{h}) &= \sigma(\sum_j w_{ij}h_j + a_i) \\
P(h_j|\mathbf{x}) &= \sigma(\sum_i w_{ij}x_i + b_j)
\end{aligned}
\tag{2.10}
$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is a sigmoid function. The gradient of the log-likelihood w.r.t the RBMs parameters is:

$$
\begin{aligned}
\nabla w_{ij} &= \langle x_i P(h_j|\mathbf{x})\rangle_0 - \langle x_i P(h_j|\mathbf{x})\rangle_\infty \\
\nabla a_i &= \langle x_i\rangle_0 - \langle x_i\rangle_\infty \\
\nabla b_j &= \langle P(h_j|\mathbf{x})\rangle_0 - \langle P(h_j|\mathbf{x})\rangle_\infty
\end{aligned}
\tag{2.11}
$$

where $\langle . \rangle_0$ represents the empirical expectation over a data distribution and $\langle . \rangle_\infty$ represents the expectation over model distribution (See Appendix B.1 for the proof). Again, we can use CD to efficiently approximate the parameters such that $\nabla w_{ij} = \langle x_i P(h_j|\mathbf{x})\rangle_0 - \langle x_i P(h_j|\mathbf{x})\rangle_k$, where k is a finite (small) number of Gibbs sampling. In many cases $k = 1$ works surprisingly well.

Despite being efficient, RBMs trained using CD with k small may not represent the data distribution accurately. Alternatively, persistent CD (PCD) [134] creates a persistent chain to sample from during the networks training. The chain is initialised randomly so that the samples are independent of the dataset. However, as expected, PCD is not as efficient as CD. In order to improve efficiency while still being able to learn a good approximation of the data distribution, the fast weight PCD [134] combines CD and PCD by running two algorithms in parallel and averaging the updates of the weights.

In case there exists label y, inference can be done through Gibbs sampling, i.e. by initially setting $y = 0.5$ and reconstructing the value of y after inferring the state of the hidden layer. Gibbs sampling is an approximation method which is necessary because of the intractable partition function. Fortunately, the conditional distribution $p(y = o|\mathbf{x})$ is tractable. For example, multiple class label can be encoded as a one-hot vector where $y = o$ is presented by setting the unit o as 1 and the other units as 0s, and the conditional distribution is computed as:

$$p(y = o|\mathbf{x}) = \frac{e^{lb_o} \prod_j (1 + e^{\mathbf{x}^\top \mathbf{w}_j + u_{oj} + b_j})}{\sum_{o'} e^{lb_{o'}} \prod_j (1 + e^{\mathbf{x}^\top \mathbf{w}_j + u_{o'j} + b_j})} \tag{2.12}$$

where $\mathbf{w}_j$ are the column vectors of the weight matrix $\mathbf{W}$ between the input layer and the hidden layer, u_{oj} are the elements in the weight matrix $\mathbf{U}$ between the label layer and the hidden layer, b_j and lb_o are biases for the units in the hidden layer and the label layer respectively.

2.2.4 Deep Belief Networks

A DBN is constructed by stacking several RBMs one on top of another [54]. The stacking is necessitated because an RBM may not be able to learn the data distribution, but an improvement can be achieved by adding one or more RBMs on top of it so that the latent variables of each RBM become the input variables of the next RBM in the stack [95, 54]. Learning can be done in sequence, i.e by training each RBM from the bottom up one at a time. This is called "unsupervised layer-wise learning" [54, 11]. Although there is no guarantee that this process will produce the improvement in learning performance mentioned above, the use of a stack of RBMs with different sizes of hidden layers trained by Contrastive Divergence has been shown useful at learning hierarchical representations [79, 78, 91, 73, 38] or for initialising a classifier [54, 11, 119].

One of the most interesting characteristics of unsupervised layer-wise learning is that it can learn different levels of representation [77, 78, 79, 120]. For example, Figure 2.4 shows a DBN trained on the MNIST handwritten digits dataset. For each hidden unit, a visualisation can be generated by setting the unit to 1 while setting all the other units in the same layer to 0, and performing downward inference to the bottom (input) layer. In this example, the different levels of representation can be visualised: in the first hidden layer, the units tend to capture low level, local information such as curves. In the second hidden layer, the units capture a higher-level of abstractions such as shapes. And finally, in the third hidden layer the units seem to represent the highest level of abstraction, i.e. the classes of the digits in the dataset.

After layer-wise training one can apply bottom-up inference to infer the state of the top layer which may include the label. One can also use the higher-level features as input to train a classifier [111, 79, 78, 73] or apply fine-tuning [54, 11, 119]. In this thesis, fine-tuning is not used because we are interested in investigating the modular unsupervised training of the networks. Instead, the raw input data is mapped onto the values of higher-level features (e.g. the values of the neurons in the third hidden layer of Figure 2.4). Such features are then provided separately as input to a classifier, together with labels, for the purpose of supervised learning; in our experiments, Support Vector Machines (SVMs) are used as classifier[2]. Furthermore, in order to evaluate the rule inference we will compare it with bottom-up inference in DBNs (§4.3).

2.3 Applications

An interesting property of deep networks is the unsupervised modular learning illustrated in Figure 2.4. In this section, we review a number of applications of this approach in different domains. More information about these applications can be found in Appendix A.

We have applied an RBM to learn features that can improve standard music similarity models (see §A.1 for more details). After training, the probability distribution of the hidden layer given a state of the visible layer, is used as input to an

[2]We use SVMs because it has a few number of hyper-parameters which is convenient for model selection

SVM for classification [125]. The use of the features produces a better classification performance in the SVM than the original data made of audio signals and texts (user tagging). The features produced by the RBM also produce a better classification performance than standard PCA features [105].

In the case of audio data, RBMs also outperform handcrafted features such as Mel-frequency Cepstral Coefficients (MFCC) [145] in a speaker recognition task (see §A.3 for more details). If we combine the features from different layers of a DBN and MFCC features the performance of the classifier can be further improved.

Features from RBMs can also be learned to help improve an action recognition task, as detailed in §A.2. The filter bases learned from the motion-difference Weizmann dataset and KTH dataset are visualised in Figure 2.5. Differently from other approaches which seek to learn local Gabor filters from the image frames [142, 74], RBMs seem to learn movement patterns as visualised as pairs of black and white lines and curves. In addition, the use of RBMs offers an improvement of prediction accuracy and learning efficiency.

Similar results can also be achieved from audio signals, for example in an application of RBMs to a music genre classification task [4]. In this application, based on the idea of convolution in unsupervised learning [78], Figure 2.6 shows representations obtained from RBMs trained on the spectrograms of different types of music. Each sub-figure corresponds to a spectrogram in a specific duration of time characterising six music genres.

More detail about applications of RBMs can be found in Appendix A. Our purpose here has been to illustrate that RBMs and stacks of RBMs can be effective at learning features and hierarchies of features without fine-tuning. It is our intention to evaluate, through the use of confidence rules and hierarchical reasoning, whether symbolic knowledge extraction can capture such hierarchies effectively. In the case of image domains, the above visualisations can be very revealing, but in the general case, we are interested in specifying explicitly how features are combined to derive new features through the application of reasoning to the confidence rules extracted from the RBMs in a modular way (i.e. one RBM at a time).

2.4 Knowledge Extraction

We have seen that representation learning can be effective in different domains. This indicates a promising use for knowledge extraction and sharing from RBMs and DBNs. The study of knowledge extraction from such networks is new. As far as we know, we have been the first to propose this challenge [126, 138] building on the work of others on knowledge extraction from Hopfield networks, Boltzmann machines and Recurrent Temporal RBMs [109, 36]. In this Section, we review several knowledge extraction techniques including the above, and discuss how they differ from our proposal.

A feed-forward neural network (NN) is a multi-layer connectionist system [114, 87, 5], which is different from DBNs, as the term is used in this thesis. The NNs are representations of input-output mapping functions while DBNs are representations of joint distributions. Research has shown that using a layer-wise learning for parameter initialisation in NNs can help achieve better performance in NNs [55, 54, 11]. Nevertheless, as already mentioned, in this thesis we are concerned with layer-wise unsupervised learning.

Extraction of symbolic knowledge from neural networks is critical for neural-symbolic integration [32, 48], where the common approach is to learn from data and background knowledge using NNs and to extract a revised symbolic knowledge from the trained networks. Most of the work on knowledge extraction has been focused on extraction algorithms applicable to neural networks trained using supervised learning, notably back-propagation. Towell and Shavlik [136] propose the M-of-N (MofN) method of rule extraction from a trained neural network. A MofN rule is expressed as $\mathsf{h} \leftarrow \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge ... \wedge \mathsf{x}_N$, meaning *"If any M out of N propositions in the body of the rule are true then* h *is true"*. For example, with $\mathsf{h} \leftarrow \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \mathsf{x}_3$, the 2of3 rule means any assignment $\{\mathsf{x}_1, \mathsf{x}_2\}$, $\{\mathsf{x}_2, \mathsf{x}_3\}$ or $\{\mathsf{x}_1, \mathsf{x}_3\}$ can imply h. Another notable work from Garcez et. al. [31] uses partially ordered sets for pruning and simplifying the extraction process. It has been shown that this method is sound (i.e. the rules extracted can be shown to approximate the function learned by the network). Similarly to [136], this approach has been evaluated on relatively small discrete datasets [136, 31].

In the era of Big Data, knowledge extraction from large networks has become important to neural-symbolic integration. Many works on knowledge extraction from text data have been proposed [2, 22, 3]. However, these approaches use either ontologies or first-order logic to represent the knowledge extracted directly from the text data. This is different from our approach where knowledge is to be extracted from the models learned without the support of background theories such as *categories* or *relations* used e.g. by statistical relational learning or inductive logic programming approaches [113, 92].

Less attention has been paid to vision data; it is difficult to interpret visual patterns with symbolic knowledge. Recent research on extracting visual sentiment relies strongly on the context, i.e. the text information associated with the images [25]. Therefore, extraction in this context is a combination of mapping visual objects to a context and building the relational knowledge that represents this context. Differently from this, in this thesis we study extraction at the pixel level without using context information.

2.5 Knowledge-Based Neural Networks

2.5.1 In-domain Symbolic Knowledge

Considerable research has been devoted to the integration of symbolic knowledge and connectionist systems [16, 27, 106, 136, 31, 32, 33, 48, 109]. The first reason for this is that symbolic rules can represent knowledge in a formal language. 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 sub-areas of Artificial Intelligence, e.g. knowledge-based transfer learning [7].

In several circumstances, prior knowledge can be provided by domain experts in the form of symbolic rules. This *in-domain knowledge* can help to improve the learning in a system. In [137], the authors propose a model named KBANN (Knowledge-Based Artificial Neural Networks) based on multi-layer feed-forward

Neural Networks to encode and learn knowledge from datasets given symbolic prior knowledge. Also using Neural Networks for knowledge representation and learning, in [8] the authors develop CIL^{2}P (Connectionist Inductive Logic Programming). The main difference between CIL^{2}P and KBANN is that CIL^{2}P uses a two-layered recurrent Neural Network [104, 115] for representing background knowledge.

Statistical models such as Markov networks and recurrent temporal restricted Boltzmann machines also have been used for neural-symbolic integration. In [113], the authors present a method to encode background knowledge into a template Markov network (named Markov Logic network (MLN)), which can be used to generate a ground Markov network to represent domain relationships from all possible instances in a dataset. The idea of representing each formula into a clique of Markov network is similar to extracting Penalty formulas in [109]. The difference is that in MLNs a feature is defined as the number of true grounding formulas corresponding to a clique in a template model, while in Penalty Logic [109] a feature is given by multiplication of variables in this clique. In practice, MLNs work successfully in a variety of relational domains. However, the model is not as comprehensible as should be expected from a symbolic model due to the fact that the number of groundings can be very large. A recent development in neural-symbolic integration is the neural-symbolic cognitive agent (NSCA) [106] in which a model based on the recurrent temporal restricted Boltzmann machines [131] is designed to represent temporal knowledge for online learning and reasoning. NSCA learns and extracts temporal rules by sampling, and it has been applied successfully to the application of driving assessment. Differently from NSCA, in our approach we are concerned with the modular representation of hierarchical knowledge applied to DBNs, and with performing hierarchical reasoning symbolically, rather than through sampling.

2.5.2 Cross-domain Knowledge: Transfer Learning

Obtaining symbolic background knowledge from a complex domain can be challenging, requiring considerable manual effort from domain experts. An alternative is to learn such knowledge from data as described above in the case of neural-symbolic integration. Even better, one could try and reuse knowledge that has been already learned and extracted from a domain to improve learning in related domains, like humans do.

Traditionally, a learning model is designed to learn from a known training dataset (*source data*) and use the learned knowledge to perform reasoning within a test dataset (*target data*) from the same distribution. Learning becomes much less effective when the test data is from a different feature space or different distribution, which can be common in real applications. In practice, collecting and labelling data is costly. Especially, when moving from one target domain to another target domain, we have to recollect and label data from that new domain to train an effective model. In many cases, labelled data from a target domain is insufficient to train the model, while one can find data from similar domains which might be easier to obtain. Thus, the idea of transfer learning is to improve the performance of machine learning applications by making use of knowledge or data from related domains.

According to [101, 133], transfer learning can be categorised into: *Inductive Transfer Learning*, *Transductive Transfer Learning*, and *Unsupervised Transfer Learning* depending on the relationship between the data and the task in the source domain and the data and the task in the target domain. In *Inductive Transfer Learning*, the task in the source domain is different from the task in the target domain while the data in the two domains can be the same. *Transductive Transfer Learning* applies to two domains with different data but the same task. Finally, *Unsupervised Transfer Learning* is similar to *Transductive Transfer Learning* with the difference that the labels in the target domain are not available.

Given the datasets from source domain and target domain, there are several options to choose when it comes to which knowledge to transfer.

First, it is possible to transfer the source data itself to the target domain [30, 29, 63]. To be more specific, the data in the source domain can be selectively trained together with data from the target domain. Here, the focus is on what parts of the data to transfer to improve performance or accuracy of the target task.

Second, one may be interested in transferring features or representations learned from the source domain to the target domain [110, 28, 81, 7]. This approach focuses on how to learn good features for the target domain with the help of source domain data. It can be done by either learning common features between source and target domain [7] or using source domain data to learn basic functions which then will be used to learn features in the target domain [110].

The third form of knowledge that can be transferred are the parameters learned [72, 15, 39, 42] so that learning in the source domain and target domain is constrained by shared parameters with the goal of improving accuracy of the targets task.

Lastly, symbolic knowledge can be transferred between the domains [89, 90, 34]. Such methods are based on the assumption that if some domains are related they should have some common relationships which can be expressed symbolically. It is expected that (parts of the) symbolic knowledge learned in the source domain will potentially help the system to discover relevant relationships in the target domain. A simple example of this form of transfer would be the learning of the definitions of plus and minus given in the previous chapter.

In this thesis, we propose a ranking of confidence rules extracted from a source RBM, which will be shown useful for selecting rules for transferring onto a target RBM. The transferring of such rules will be shown capable of improving accuracy in the target domain, without the need for transferring extensive source domain data.

2.6 Summary

This chapter reviewed the background literature of the thesis. We revised the theories and applications of energy-based unsupervised learning models. Hopfield networks, BMs, RBMs and DBNs have been studied. After that we discussed the promising of such models in knowledge extraction, neural-symbolic integration and transfer learning. The focus of this thesis, as mentioned in this chapter, is on unsupervised layer-wise approach to understand the effectiveness of modular learning and reasoning, inspired by the work of Penalty Logic about symbolic representation and reasoning in energy-based connectionist systems.

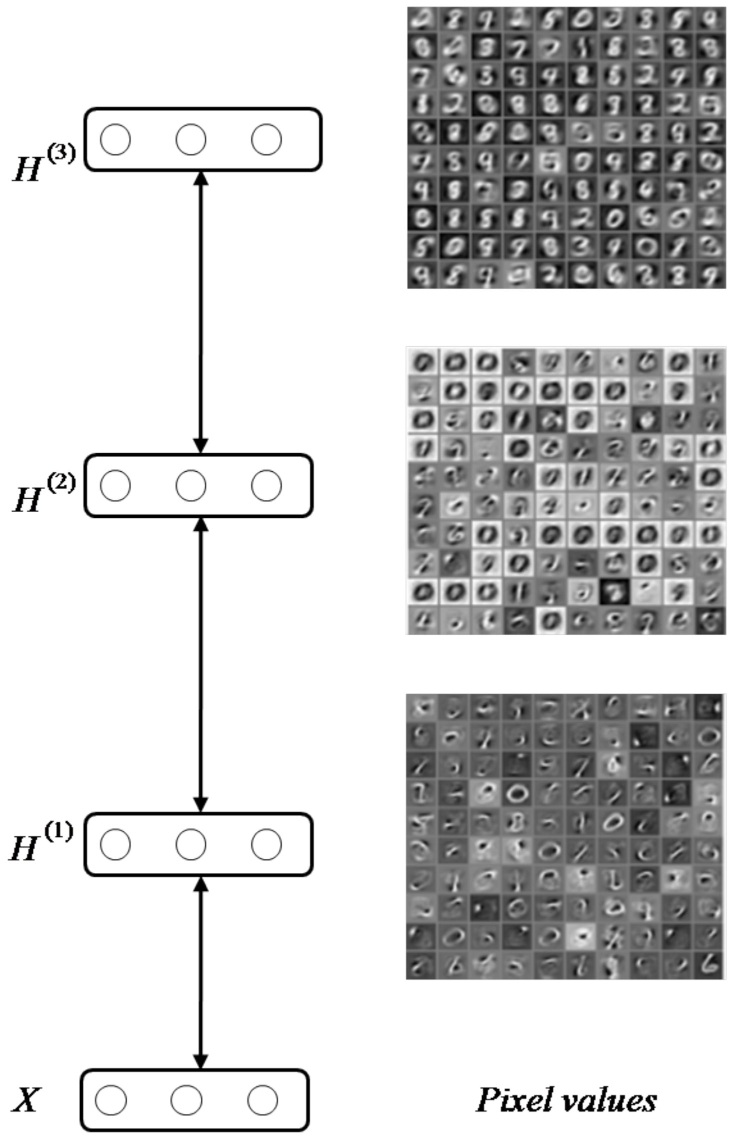
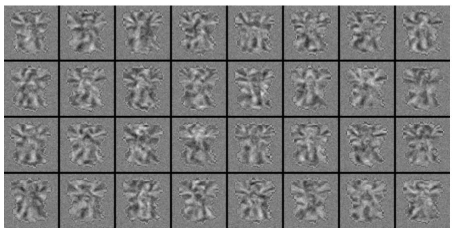


Figure 2.4: A stack of RBMs trained on handwritten digits with visualisations of the units in the hidden layers obtained by activating each hidden unit at a time and performing downward inference to the visible layer thus generating pixels for the images. The hierarchy is expected to model levels of abstraction by transforming the feature vectors, in this case from edges to shapes and finally digits.


(a) Weizmman dataset.

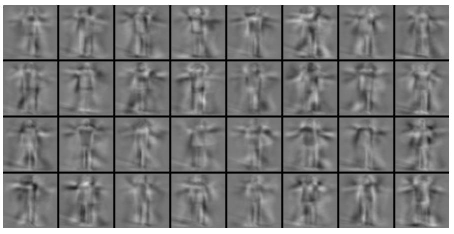
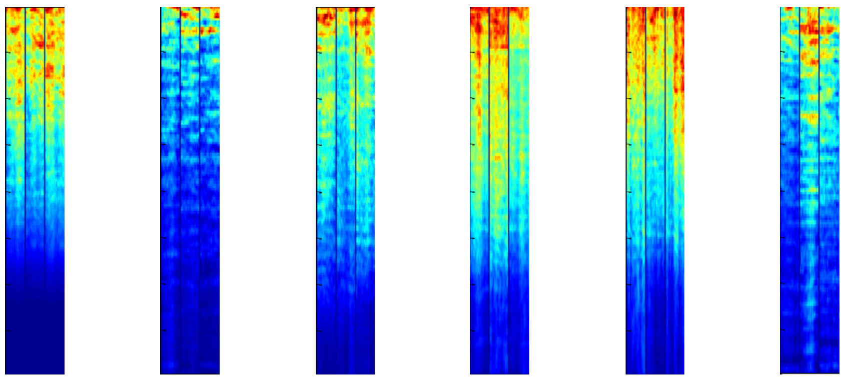

(b) KTH dataset.

Figure 2.5: Visualisation of 24 filter bases from RBMs trained on motion-difference of (a) Weizmann and (b) KTH datasets.



(a) Blues. (b) Classical. (c) Country. (d) Disco. (e) Hiphop. (f) Jazz.

Figure 2.6: Visualisation of the filter bases of a Gaussian RBM trained on spectrograms of different types of music.

Chapter 3

Propositional Calculus and Deep Belief Networks

In this chapter we take a closer look at the related work which establishes a relationship between propositional logic and DBNs. Propositional calculus has been shown equivalent to minimising an energy function. We introduce confidence rules formally as if-and-only-if formulas in propositional logic which are, in addition, each associated with a real value, called "confidence value". Confidence rules are shown to guarantee that the logical models of a formula (true or false assignments which map the formula to true) will have minimum energy in the corresponding DBN. We then show that confidence rules can be used to approximate an unknown set of models trained in a DBN. This suggests that it is possible to extract knowledge from trained DBNs in the form of confidence rules, which we will study further in the next chapter.

3.1 Propositional Calculus and Energy-based Neural Networks

3.1.1 Propositional Logic

In propositional logic, an atomic proposition is a statement or assertion which can only be *true* or *false*. One can use a symbol to represent an atomic proposition, such as: r for "it is raining". A well-formed formula (WFF) is constructed by combining atomic propositions using the connectives: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\leftarrow$ (IF-THEN), $\leftrightarrow$ (IF-AND-ONLY-IF). For example:

$$\mathsf{u} \leftarrow ((\mathsf{x} \wedge \neg \mathsf{y}) \vee (\neg \mathsf{x} \wedge \mathsf{y}) \leftrightarrow \mathsf{z})$$

where $\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{u}$ are propositions. We can also represent a well-formed formula as a combination of other well-formed formulas (called sub-formulas). It is convenient sometimes to define notation such as $\mathsf{x} \oplus \mathsf{y}$ used to denote the Exclusive-Or operator between two sub-formulas. The $\oplus$ operator is shorthand for using the connectives to express that the outcome is true if and only if the truth-values of the sub-formulas are different, that is: $(\neg \mathsf{x} \wedge \mathsf{y}) \vee (\mathsf{x} \wedge \neg \mathsf{y})$.

Given a truth-value assignment to a WFF which maps each atomic proposition to a truth-value true or false, we can decide the truth-value of the formula. Let us use $s_{\varphi}(\mathbf{x})$ to denote the truth-value of a WFF φ which consists of $\mathbf{x} = \{x_i | i = 1, .., I\}$ atomic propositions. An assignment is called a model or "preferred assignment" of a formula if and only if the formula is *true* with this assignment.

3.1.2 Penalty Logic

The first work to study the symbolic representation of energy-based neural networks was Penalty logic [109]. Penalty logic is an extension of propositional logic where a penalty logic well-formed formula (PLOFF) is defined as a finite set of pairs (ρ, φ), in which each WFF φ is associated with a real value ρ called *penalty*.

Given a truth-value assignment, a PLOFF is evaluated by a ranking function

V_{rank} **which calculates the sum of all the penalties from the PLOFFs assigned to** $false$. A preferred assignment $\mathbf{x}$ is then an assignment with a lowest total penalty. Applied to classification, for example, to decide the truth-value of a target proposition y given an assignment x of the other propositions, one will choose the value of y that has the lowest $V_{rank}(\mathsf{x}, \mathsf{y})$.

Example 3.1.1. Suppose we are given a set of PLOFFs as below:

$$(1, \neg\mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z})$$
$$(1, \mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z})$$
$$(1, \mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z})$$
$$(1, \neg\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z})$$

Given $\mathsf{x} = true$ and $\mathsf{y} = false$ we have $V_{rank}(\mathsf{x} = true, \mathsf{y} = false, \mathsf{z} = false) = 4$ and $V_{rank}(\mathsf{x} = true, \mathsf{y} = false, \mathsf{z} = true) = 3$, so one should conclude that $\mathsf{z} = true$.

3.1.3 Penalty Logic and Boltzmann Machines

A PLOFF can be represented by an energy-based neural network [109]. It has been shown that reasoning with ranking function in Penalty logic is equivalent to minimising energy function in a Boltzmann machine [107, 108, 109]. The idea is to convert all PLOFFs into a Boltzmann machine with an energy E function such that:

$$V_{rank} = E_{rank} + constant$$

where $E_{rank}(\mathbf{x}) = min_{\mathbf{h}} E(\mathbf{x}, \mathbf{h})$ is the energy function minimised over all hidden variables. The equivalent Boltzmann machine will have binary visible units capturing the truth-values of propositions x, with Boolean values $false$, $true$ represented by 0, 1, respectively. The details of this conversion have been discussed in [109]; in what follows we show an example as a case in point.

Example 3.1.2. The XOR formula: $(1, \mathsf{x} \oplus \mathsf{y} \leftrightarrow \mathsf{z})$ can be represented by an energy function $E = 2xy - 2xz - 2yz - 8xh - 8yh + 8zh + x + y + z + 12h$, where h is an additional hidden variable (see Appendix B.2 for more details). Notice that $\mathsf{x}, \mathsf{y}, \mathsf{z}$ are the propositional logic representation of the binary variables x, y, z such that the $true, false$ assignments to the propositions $(\mathsf{x}, \mathsf{y}, \mathsf{z})$ are equivalent to the assignment

x	y	z	$E(x,y,z,h=0)$	$E(x,y,z,h=1)$	$E_{rank}(x,y,z)$	$\mathsf{x} \oplus \mathsf{y} \leftrightarrow \mathsf{z}$	$V_{rank}(\mathsf{x} \oplus \mathsf{y} \leftrightarrow \mathsf{z})$
0	0	0	0	12	**0**	true	**0**
0	0	1	1	21	**1**	false	**1**
0	1	0	1	5	**1**	false	**1**
0	1	1	0	12	**0**	true	**0**
1	0	0	1	5	**1**	false	**1**
1	0	1	0	12	**0**	true	**0**
1	1	0	4	0	**0**	true	**0**
1	1	1	1	5	**1**	false	**1**

Table 3.1: XOR formula and Penalty logic ranking.

of $1, 0$, respectively, to the variables (x, y, z). Table 3.1 shows that the truth-values of $\mathsf{x}, \mathsf{y}, \mathsf{z}$ which assign *true* to the formula also achieve minimum energy (E_{rank}). With the energy function above we can represent the XOR formula in an energy-based neural network, as shown in Table 3.1.

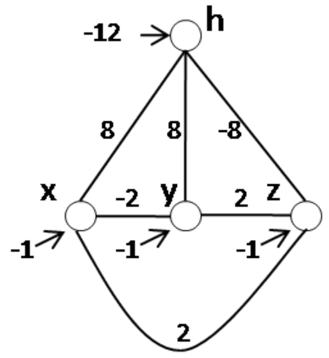


Figure 3.1: A Boltzmann machine for the XOR formula.

Knowledge extraction using Penalty Logic can be done by converting an energy-based function into PLOFF [109]. First, all hidden units need to be eliminated before translating every product term in energy functions into a conjunction and the penalty will be the real value factor in that term, i.e. $15xyz$ is translated to $(15, \mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z})$. However, the complexity of elimination process to remove one single hidden unit in an energy function is exponential on the number of visible units connecting to it [109]. This makes the extraction of knowledge from a complex domain intractable.

3.1.4 Penalty Logic and RBMs

Knowledge extraction of Penalty Logic rules is difficult when there are hidden units in the network and one wants to extract rules for the visible units only. Extracting Penalty Logic rules that contain logical propositions for the hidden units is possible, but one then needs to ask what the meaning of such hidden units might be.

In RBMs the hidden units are assumed to capture levels of abstraction by representing independent feature detectors. Therefore, knowledge extraction from RBMs may be easier to interpret even with the existence of hidden units. However, extracting Penalty Logic rules without elimination of hidden units from RBMs results in a set of relations of the form: (visible unit, hidden unit), when we are in fact more interested in the relations between the visible units. For example, given an RBM with the energy function $E = -5xh + 3yh + 4zh$, the Penalty Logic rules extracted are: $(5, \neg(\mathsf{x} \wedge \mathsf{h})), (3, \mathsf{y} \wedge \mathsf{h}), (4, \mathsf{z} \wedge \mathsf{h})$[1], which would have to be manipulated algebraically following extraction to reveal the relationships between x, y, z.

3.2 Propositional Calculus and RBMs

We have reviewed the idea of Penalty Logic as an extension of propositional logic and discussed the equivalence of Penalty Logic and Boltzmann machines, as proved by Pinkas [109]. We have seen that it can be difficult to extract knowledge from RBMs using Penalty Logic. In this section, we show that it is possible to represent a WFF in an RBM and that each hidden unit of an RBM captures a preferred assignment of the WFF. To this end, we convert WFFs into disjunctive normal form (as detailed below) instead of conjunctions of sub-formulas as in Penalty Logic [108, 109].

In propositional logic, any WFF can be represented in disjunctive normal form (DNF) [117]:

$$\varphi = \bigvee_j (\bigwedge_{t \in T_j} \mathsf{x}_t \wedge \bigwedge_{k \in K_j} \neg \mathsf{x}_k)$$

where each $(\bigwedge_{t \in T_j} \mathsf{x}_t \wedge \bigwedge_{k \in K_j} \neg \mathsf{x}_k)$ is called a "nested conjunction".

[1]Theorem 4.13 in [109] with h is treated as same as visible variables

Definition 3.2.1.

- *A "strict DNF" (SDNF) is a DNF where a single nested conjunction is true.*
- *A "full DNF" is a DNF with each variable appearing once in every nested conjunction.*

Example 3.2.1. The XOR formula: $\varphi = (\mathsf{x} \oplus \mathsf{y}) \leftrightarrow \mathsf{z}$ has its truth-table shown in Table 3.2.

x	y	z	φ
false	*false*	*false*	*true*
false	*false*	*true*	*false*
false	*true*	*false*	*false*
false	*true*	*true*	*true*
true	*false*	*false*	*false*
true	*false*	*true*	*true*
true	*true*	*false*	*true*
true	*true*	*true*	*false*

Table 3.2: Truth table of XOR formula: $(\mathsf{x} \oplus \mathsf{y}) \leftrightarrow \mathsf{z}$.

For each assignment that returns *true* for the formula, for example $\{\mathsf{x} = true, \mathsf{y} = true, \mathsf{z} = false\}$ we create a nested conjunction $\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z}$. The XOR formula can therefore be converted into a full DNF:

$$\varphi = (\neg\mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z}) \vee (\neg\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z})$$

Example 3.2.2. Given a DNF $\varphi = (\mathsf{x} \wedge \mathsf{y}) \ \vee (\mathsf{x} \wedge \mathsf{z}) \vee (\neg\mathsf{x})$, a preferred assignment $\mathsf{x} = true, \mathsf{y} = true, \mathsf{z} = true$ makes both $(\mathsf{x} \wedge \mathsf{y})$ and $(\mathsf{x} \wedge \mathsf{z})$ *true*. The sub-formula $(\mathsf{x} \wedge \mathsf{y}) \ \vee (\mathsf{x} \wedge \mathsf{z})$ can be replaced by a full DNF $(\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z}) \vee (\mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z})$ such that the original WFF φ becomes $\varphi = (\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z}) \vee (\mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z}) \vee (\neg\mathsf{x})$ which is a SDNF. Notice how part of the WFF is converted into a full DNF which is combined with the remaining part of the WFF to form a SDNF.

Definition 3.2.2. *A WFF φ is said to correspond to an energy-based network N if and only if for a truth-value assignment $\mathbf{x}$, $s_\varphi(\mathbf{x}) = -AE_{Nrank}(\mathbf{x}) + B$, where $s_\varphi(\mathbf{x})$ is the set of positive propositions in $\mathbf{x}$, $A > 0$, B is a fixed real number, and $E_{Nrank}(\mathbf{x}) = min_{\mathbf{h}} E_N(\mathbf{x}, \mathbf{h})$ is the energy function of N minimised over all hidden units.*

The above correspondence guarantees that all preferred assignments of a WFF correspond to a minimum in the energy function of the network. In addition, by construction, all assignments of the formula to *false* correspond to a maximum of the energy function. This means that the network must have only two states of energy which correspond to *false*, *true* in the formula.

Lemma 3.2.1. *Any SDNF φ can be mapped onto a corresponding energy-based network N with energy function $E = -\sum_j \prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k)$ where T_j, K_j are respectively the sets of positive and negative propositions of each nested conjunction j in the SDNF.*

Proof. By definition, $\varphi = \bigvee_j(\bigwedge_{t\in T_j} \mathsf{x}_t \wedge \bigwedge_{k\in K_j} \neg\mathsf{x}_k)$. Each nested conjunction $\bigwedge_{t\in T_j} \mathsf{x}_t \wedge \bigwedge_{k\in K_j} \neg\mathsf{x}_k$ corresponds to $\prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k)$ which maps to 1 if and only if $x_t = 1$ ($\mathsf{x}_t = true$) and $x_k = 0$ ($\mathsf{x}_k = false$) for all $t \in T_j$ and $k \in K_j$. Since φ is a SDNF, that is *true* if and only if one nested conjunction is *true*, then the sum $\sum_j \prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k) = 1$ if and only if the assignment of truth-values for x_t , x_k is a preferred assignment of φ. Hence, there exists an energy-based network N with energy function $E = -\sum_j \prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k)$ such that $s_\varphi(\mathbf{x}) = -E_N(\mathbf{x})$. □

x	y	z	s_φ(x, y, z)	$E_N(x,y,z)$
false	*false*	*false*	*true*	−1
false	*false*	*true*	*false*	0
false	*true*	*false*	*false*	0
false	*true*	*true*	*true*	−1
true	*false*	*false*	*false*	0
true	*false*	*true*	*true*	−1
true	*true*	*false*	*true*	−1
true	*true*	*true*	*false*	0

Table 3.3: Energy function and truth table of XOR formula.

Example 3.2.3. Example 3.2.1 showed that the XOR formula can be converted into a SDNF as:

$$\varphi = (\neg\mathsf{x} \wedge \neg\mathsf{y} \wedge \neg\mathsf{z}) \vee (\neg\mathsf{x} \wedge \mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z}) \vee (\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z})$$

For each nested conjunction, for example $\mathsf{x} \wedge \mathsf{y} \wedge \neg\mathsf{z}$ we create a term $xy(1-z)$ and add it to the energy function. After all terms are added, we have the energy function for N:

$$E(x,y,z) = -(1-x)(1-y)(1-z) - xy(1-z) - x(1-y)z - (1-x)yz$$

The correspondence between this energy function and the truth-values of the formula is illustrated in Table 3.3.

The correspondence between φ and N exists because $s_{\varphi}(\mathsf{x},\mathsf{y},\mathsf{z}) = -E_N(x,y,z)$. If we expand the energy function above we can see it is equivalent to the energy function used by Penalty Logic minus one (see Example 3.1.2). Note that we use a different method to generate the energy function (see Appendix B.2 for more details on how energy functions are generated in Penalty Logic).

We have seen that any SDNF φ can be mapped onto energy function $E = -\sum_j \prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k)$. Let us denote $|T_j|$ as the number of positive propositions in a nested conjunction j. For each term $e_j(\mathbf{x}) = -\prod_{t\in T_j} x_t \prod_{k\in K_j}(1-x_k)$ we can construct another energy function with an additional hidden variable h_j as: $\tilde{e}_j(\mathbf{x},h_j) = h_j(|T_j| - \sum_{t\in T_j} x_t + \sum_{k\in K_j} x_k - \epsilon)$ with $0<\epsilon<1$ such that $e_j(\mathbf{x}) = \frac{1}{\epsilon}\tilde{e}_{j\,rank}(\mathbf{x})$. This correspondence holds because $|T_j| - \sum_t x_t + \sum_k x_k - \epsilon = -\epsilon$ if and only if $x_t = 1$ and $x_k = 0$ for all $t\in T_j$ and $k \in K_j$ that make $min_{h_j}\tilde{e}_j(\mathbf{x},h_j) = -\epsilon$ with $h_j = 1$. Otherwise, the result is larger than zero and then $min_{h_j}\tilde{e}_j(\mathbf{x},h_j) = 0$ with $h_j = 0$.

We can conclude that the energy function for a set of SDNFs becomes:

$$\tilde{E}(\mathbf{x}) = -\sum_j h_j(\sum_t x_t - \sum_k x_k - |T_j| + \epsilon) \tag{3.1}$$

which is an energy function of an RBM. Note that $s_{\theta}(\mathbf{x}) = -\frac{1}{\epsilon}\tilde{E}_{rank}(\mathbf{x})$

Construction 3.2.1. *An RBM can be constructed from a WFF as follows:*

- *Convert a WFF into SDNF.*
- *For all nested conjunctions j:* $\bigwedge_{t\in T_j} \mathsf{x}_t \wedge \bigwedge_{k\in K_j} \neg\mathsf{x}_k$.
  - *Create a hidden unit* h_j.
  - *Create a connection between all visible units t* ($t \in T_j$) *and the hidden unit j with a weight* $w_{tj} = 1$.
  - *Create a connection between all visible units k* ($k \in K_j$) *and the hidden unit j with a weight* $w_{kj} = -1$.
  - *Set the bias* $b_j = -|T_j| + \epsilon$ *with* $0 < \epsilon < 1$ *for the hidden unit j.*

Example 3.2.4. In Example 3.2.1 the XOR formula can be converted into:

$$\varphi = (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z)$$

We can then construct the RBM of Figure 3.2. In this example we choose $\epsilon = 0.5$.

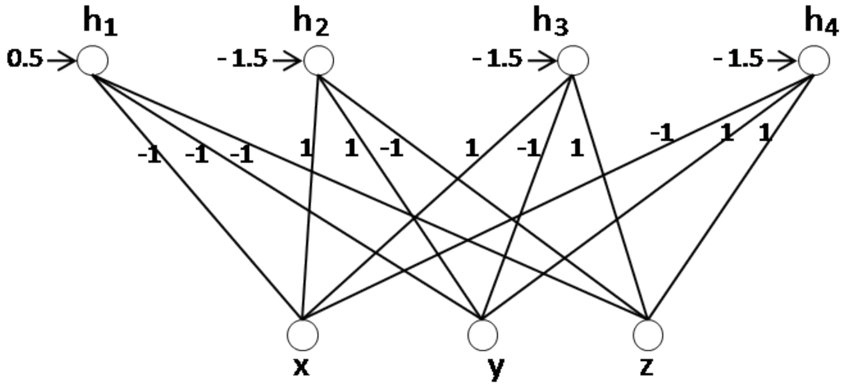


Figure 3.2: RBM for XOR formula: $(x \oplus y) \leftrightarrow z$.

The energy function of this RBM is:

$$E(x,y,z) = xh_1 + yh_1 + zh_1 - xh_2 - yh_2 + zh_2 - xh_3 + yh_3 + -zh_3 + xh_4 - yh_4 - zh_4 \\ - 0.5h_1 + 1.5h_2 + 1.5h_3 + 1.5h_4$$

and the correspondence between the formula and RBM is illustrated in Table 3.4.

x	y	z	$s_\varphi(x,y,z)$	$E_{Nrank}(x,y,z)$
false	*false*	*false*	*true*	−0.5
false	*false*	*true*	*false*	0
false	*true*	*false*	*false*	0
false	*true*	*true*	*true*	−0.5
true	*false*	*false*	*false*	0
true	*false*	*true*	*true*	−0.5
true	*true*	*false*	*true*	−0.5
true	*true*	*true*	*false*	0

Table 3.4: Minimised energy function of RBM representing truth-table of XOR formula.

3.3 Propositional Calculus and DBNs

We have seen how a propositional formula can be encoded into a corresponding RBM. In this section, we investigate how a set of formulas can be encoded in a DBN by stacking multiple RBMs. First we show how to construct a DBN from a DNF by decomposing nested conjunctions into smaller groups of shared conjunctions that include hidden variables. We will see that such DBNs do not correspond to the original formula and therefore confidence rules will have to be introduced to re-establish correspondence.

3.3.1 Decomposition and Stacking

In a nested conjunction, it may be convenient to group some propositions together with the use of a new hidden variable. For example: $\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg\mathsf{x}_3 \wedge \mathsf{x}_4$ may be expressed equivalently as $(\mathsf{h} \wedge \neg\mathsf{x}_3 \wedge \mathsf{x}_4) \wedge (\mathsf{h} \leftrightarrow (\mathsf{x}_1 \wedge \mathsf{x}_2))$, where h is a new hidden variable. If many nested conjunctions of a WFF share the same subset of the propositions, the above use of a hidden variable can save space and make the WFF more readable.

Example 3.3.1. A well-formed formula $\varphi = \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge (\mathsf{x}_3 \oplus \mathsf{x}_4)$ can be converted to DNF as follows [71]:

$$\begin{aligned}
\varphi &= \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge (\mathsf{x}_3 \oplus \mathsf{x}_4) \\
&= (\mathsf{x}_1 \wedge \mathsf{x}_2) \wedge ((\neg\mathsf{x}_3 \wedge \mathsf{x}_4) \vee (\mathsf{x}_3 \wedge \neg\mathsf{x}_4)) \\
&= (\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \neg\mathsf{x}_3 \wedge \mathsf{x}_4) \vee (\mathsf{x}_1 \wedge \mathsf{x}_2 \wedge \mathsf{x}_3 \wedge \neg\mathsf{x}_4)
\end{aligned}$$

Consider 3 groups of propositions $(\mathsf{x}_1, \mathsf{x}_2)$, $(\neg\mathsf{x}_3, \mathsf{x}_4)$, $(\mathsf{x}_3, \neg\mathsf{x}_4)$ from which we create three hidden propositions $\mathsf{h}_1, \mathsf{h}_2, \mathsf{h}_3$. Therefore:

$$\begin{aligned}
\varphi = {} & ((\mathsf{h}_1 \wedge \mathsf{h}_2) \vee (\mathsf{h}_1 \wedge \mathsf{h}_3)) \\
& \wedge (\mathsf{h}_1 \leftrightarrow (\mathsf{x}_1 \wedge \mathsf{x}_2)) \\
& \wedge (\mathsf{h}_2 \leftrightarrow (\neg\mathsf{x}_3 \wedge \mathsf{x}_4)) \\
& \wedge (\mathsf{h}_3 \leftrightarrow (\mathsf{x}_3 \wedge \neg\mathsf{x}_4))
\end{aligned}$$

Definition 3.3.1. *A conjunctive if-and-only-if formula (CIFF) is an if-and-only-if formula* *with a single hidden variable in one side and a conjunction of propositions in the other side of the biconditional connective, as follows:*

$$\mathsf{h} \leftrightarrow \bigwedge_t \mathsf{x}_t \wedge \bigwedge_k \neg \mathsf{x}_k$$

Given an assignment of truth-values to x_t, x_k, the truth-value of h will be known and will be the same as the truth-value of $\bigwedge_t \mathsf{x}_t \wedge \bigwedge_k \neg \mathsf{x}_k$.

A set of CIFFs can be encoded into an RBM where each hidden unit represents a hidden variable, as exemplified below.

Example 3.3.2. The three CIFFs $\mathsf{h}_1 \leftrightarrow (\mathsf{x}_1 \wedge \mathsf{x}_2)$, $\mathsf{h}_2 \leftrightarrow (\neg \mathsf{x}_3 \wedge \mathsf{x}_4)$, $\mathsf{h}_3 \leftrightarrow (\mathsf{x}_3 \wedge \neg \mathsf{x}_4)$ from Example 3.3.1 can be encoded in the RBM of Figure 3.3.

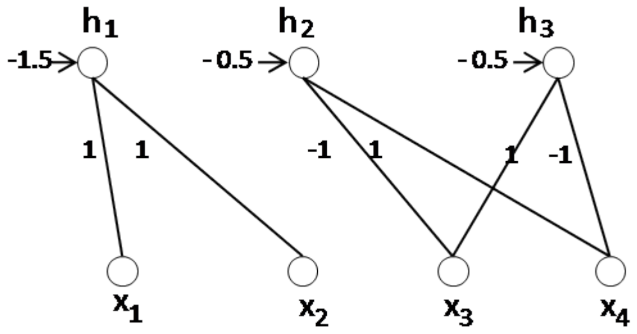


Figure 3.3: RBM for CIFFs.

Construction 3.3.1. *One can construct a DBN given a formula as follows. First, a WFF can be converted into a DNF. After that we find common propositions in its nested conjunctions to add hidden variables as shown in Example 3.3.1. This allows the creation of an RBM as illustrated earlier. It is possible to repeat the process, converting the formula with hidden variables into DNF (or SDNF in the case of the top layer) to create another RBM to go on top of the previous one. This hierarchical organisation of symbolic knowledge will be shown useful in practice in the next chapters.*

Example 3.3.3. Consider the WFF $\varphi = \mathsf{x}_1 \wedge \mathsf{x}_2 \wedge (\mathsf{x}_3 \oplus \mathsf{x}_4)$ from Example 3.3.1. After adding three hidden variables we have built the RBM from Example 3.3.2. However, the following (higher-level) formula was left out and is now to be encoded in another RBM on top of the original one.

$$(\mathsf{h}_1 \wedge \mathsf{h}_2) \vee (\mathsf{h}_1 \wedge \mathsf{h}_3)$$

As done before, the formula can be converted into a SDNF:

$$(h_1 \wedge h_2 \wedge \neg h_3) \vee (h_1 \wedge h_2 \wedge h_3) \vee (h_1 \wedge \neg h_2 \wedge h_3)$$

Three higher-level hidden variables can be created and the process repeated to produce the DBN of Figure 3.4.

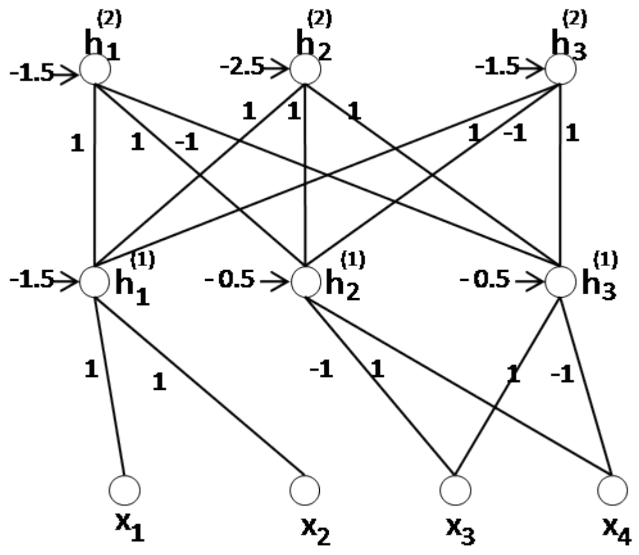


Figure 3.4: A DBN for $x_1 \wedge x_2 \wedge (x_3 \oplus x_4)$.

We have seen how to construct a DBN from a WFF. However, one can notice that this DBN does not correspond to the WFF. In Example 3.3.3, the formula will be *false* for the assignment x_1 = *true*, x_2 = *true*, x_3 = *true*, x_4 = *true*, while $E_{rank}(x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1) = -0.5$. One can also notice that the energy of the top RBM corresponds to the formula regardless of the energy of lower RBMs. This is because the top RBM is constructed from a SDNF while the lower RBMs are created from CIFFs. For example, the non-preferred assignment x_1 = *true*, x_2 = *true*, x_3 = *true*, x_4 = *true* implies that h_1 must be true given CIFF $h_1 \leftrightarrow x_1 \wedge x_2$. In order to re-establish correspondence between DBNs and WFFs in what follows we define confidence rules as an extension of CIFFs.

3.3.2 Confidence Rules

A DBN can be seen as an approximation of a hierarchy of CIFFs. We now define confidence rules as an extension of CIFFs.

Definition 3.3.2. *A confidence rule is a CIFF associated with a confidence value c, written:*

$$c : \mathsf{h} \leftrightarrow \bigwedge_t \mathsf{x}_t \wedge \bigwedge_k \neg \mathsf{x}_k \tag{3.2}$$

where c is a non-negative real number.

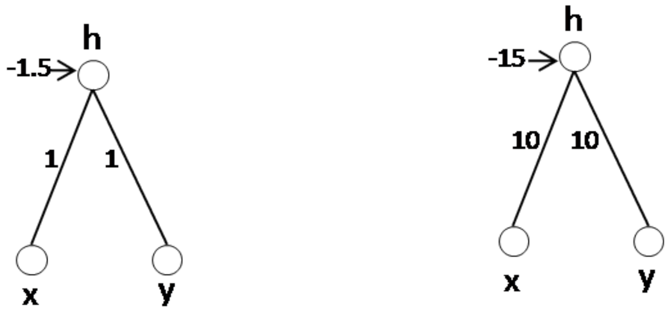
Similar to the "penalty" in [109], the term "confidence" in this definition expresses the "strength of belief", "reliability" of knowledge. The difference is, in Penalty Logic the inference engine gives preference to the assignments with lower penalties, whilst in confidence rules the preference is given to the assignments with higher confidences. In statistic, although sharing the same term "confidence", the "confidence interval" is much different from ours in that it is used to estimate a range of values. The confidence value of a rule is also different from the terms "confidence" and "confidence level" in NSCA [106] which is a probability, although our "confidence values" and their "confidence" can be used to measure the credibility of the rules. The reason we use confidence values as real values is that we can treat them as the parameters of a program, and therefore the inference can be performed as in parametric models. In particular, our confidence values link closely to the energy function of RBMs which forms the probability function for an assignment. However, note that the probability function of an RBM is intractable. Instead, using the confidences as real values would be useful for inference as we will show in the next chapter.

A confidence rule, for example $c : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg\mathsf{x}_2 \wedge \mathsf{x}_3$, can be read as "*If* x_1 *is true and* x_2 *is false then; assuming that* h *is true,* x_3 *should be true with confidence c*". We can also apply this to conclude x_1 given the truth assignments of x_2 and x_3, or to conclude $\neg\mathsf{x}_2$ given the truth assignments of x_1 and x_3 . In the case there exists a label proposition it will be located at the end of the rule. In §4.1.2 we discuss in details how confidence rules can be interpreted.

3.3.3 Confidence Rules and DBNs

The structure of a DBN constructed from a set of confidence rules is the same as that constructed from CIFFs. The difference is that the confidence values will alter the weights and biases. In particular, weights and biases will be multiplied by the confidence value. Therefore, the energy contribution of the part of the DBN constructed from a confidence rule with confidence value c will be $-c \times \epsilon$ for preferred assignments and 0 for non-preferred assignments.

Example 3.3.4. Let $10 : \mathsf{h} \leftrightarrow \mathsf{x} \wedge \mathsf{y}$ be a confidence rule for a CIFF $\mathsf{h} \leftrightarrow \mathsf{x} \wedge \mathsf{y}$ with confidence value 10. Two networks constructed from this CIFF and the confidence rule are shown in Figure 3.5. The truth-values and energy for all



(a) A network $N1$ for CIFF: $\mathsf{h} \leftrightarrow \mathsf{x} \wedge \mathsf{y}$. (b) A network $N2$ for confidence rule: $10 : h \leftrightarrow \mathsf{x} \wedge \mathsf{y}$.

Figure 3.5: Networks for CIFF ($N1$) and confidence rule ($N2$).

possible assignments are shown in Table 3.5, where $N1$ and $N2$ are the networks constructed given a CIFF and confidence rule, respectively. It is interesting noting that $s_{\mathsf{h}\leftrightarrow\mathsf{x}\wedge\mathsf{y}}(\mathsf{x},\mathsf{y}) = -\frac{1}{5}E_{N2rank}(x,y)$, $E_{N1rank}(x,y) = min_h(-xh - yh + 1.5h)$, and $E_{N2rank}(x,y) = 10E_{N1rank}(x,y)$.

x	y	$s_{\mathsf{x}\wedge\mathsf{y}}$	h	E_{N1rank}	E_{N2rank}
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	-0.5	-5
1	1	1	1	-0.5	-5

Table 3.5: Truth-table of conjunction $\mathsf{x} \wedge \mathsf{y}$ and energy of the networks constructed from CIFF $\mathsf{h} \leftrightarrow \mathsf{x} \wedge \mathsf{y}$ and confidence rule $5 : \mathsf{h} \leftrightarrow \mathsf{x} \wedge \mathsf{y}$ respectively.

Theorem 3.3.1. *Any confidence rule can be approximated by a corresponding DBN.*

Proof. We can use Construction 3.3.1 to build a DBN but converting all CIFFs into confidence rules with the same very small confidence value c_0 (close to zero). We also convert the top SDNF into confidence rules sharing a very large confidence value c_∞ (close to positive infinity). The minimum energy of the DBN will be $E_{rank}(\mathbf{x}) \approx -c_\infty\epsilon$ for preferred assignments and $E_{rank}(\mathbf{x}) = -K_{\mathbf{x}}c_0\epsilon \gg -c_\infty\epsilon$ for non-preferred assignments ($K_{\mathbf{x}}$ is the number of CIFFs that are mapped to *true* given assignment $\mathbf{x}$). As a result, $s(\mathbf{x}) \approx -\frac{1}{c_\infty\epsilon}E_{rank}(\mathbf{x})$. □

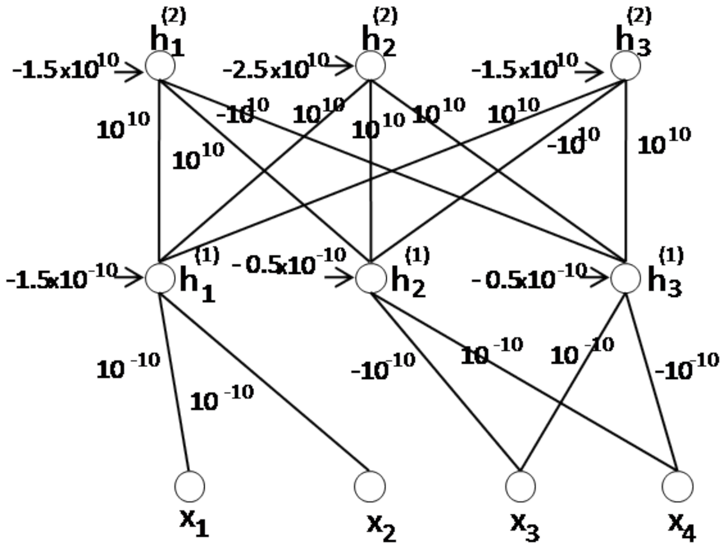


Figure 3.6: DBN corresponding to $x_1 \wedge x_2 \wedge (x_3 \oplus x_4)$.

Example 3.3.5. For the formula $(x_1 \wedge x_2) \wedge (x_3 \oplus x_4)$ in Example 3.3.1, we have seen that the DBN constructed in Example 3.3.3 does not correspond to the formula. Let us now convert the three CIFFs:

$$h_1 \leftrightarrow (x_1 \wedge x_2)$$
$$h_2 \leftrightarrow (\neg x_3 \wedge x_4)$$
$$h_3 \leftrightarrow (x_3 \wedge \neg x_4)$$

into confidence rules with small confidence values, as follows:

$$10^{-10} : h_1^{(1)} \leftrightarrow (x_1 \wedge x_2)$$
$$10^{-10} : h_2^{(1)} \leftrightarrow (\neg x_3 \wedge x_4)$$
$$10^{-10} : h_3^{(1)} \leftrightarrow (x_3 \wedge \neg x_4)$$

and the top DNF: $(\mathsf{h}_1 \wedge \mathsf{h}_2 \wedge \neg\mathsf{h}_3) \vee (\mathsf{h}_1 \wedge \mathsf{h}_2 \wedge \mathsf{h}_3) \vee (\mathsf{h}_1 \wedge \neg\mathsf{h}_2 \wedge \mathsf{h}_3)$ into:

$$10^{10} : \mathsf{h}_1^{(2)} \leftrightarrow \mathsf{h}_1^{(1)} \wedge \mathsf{h}_2^{(1)} \wedge \neg\mathsf{h}_3^{(1)}$$
$$10^{10} : \mathsf{h}_2^{(2)} \leftrightarrow \mathsf{h}_1^{(1)} \wedge \mathsf{h}_2^{(1)} \wedge \mathsf{h}_3^{(1)}$$
$$10^{10} : \mathsf{h}_3^{(2)} \leftrightarrow \mathsf{h}_1^{(1)} \wedge \neg\mathsf{h}_2^{(1)} \wedge \mathsf{h}_3^{(1)}$$

with large confidence values, producing the DBN in Figure 3.6.

Table 3.6 shows the truth-values of the formula and energy of the DBN over all possible assignments.

x_1	x_2	x_3	x_4	$s_{(\mathsf{x}_1 \wedge \mathsf{x}_2) \wedge (\mathsf{x}_3 \oplus \mathsf{x}_4)}$	E_{rank}
0	0	0	0	0	$= 0$
0	0	0	1	0	$\approx 0\ (-0.5 \times 10^{-10})$
0	0	1	0	0	$\approx 0\ (-0.5 \times 10^{-10})$
0	0	1	1	0	$= 0$
0	1	0	0	0	$= 0$
0	1	0	1	0	$\approx 0\ (-0.5 \times 10^{-10})$
0	1	1	0	0	$\approx 0\ (-0.5 \times 10^{-10})$
0	1	1	1	0	$= 0$
1	0	0	0	0	$= 0$
1	0	0	1	0	$\approx 0\ (-0.5 \times 10^{-10})$
1	0	1	0	0	$\approx 0\ (-0.5 \times 10^{-10})$
1	0	1	1	0	$= 0$
1	1	0	0	0	$\approx 0\ (-0.5 \times 10^{-10})$
1	1	0	1	1	$\approx -0.5 \times 10^{10}\ (-0.5 \times 10^{10} - 0.5 \times 10^{-10})$
1	1	1	0	1	$\approx -0.5 \times 10^{10}\ (-0.5 \times 10^{10} - 0.5 \times 10^{-10})$
1	1	1	1	0	$\approx 0\ (-0.5 \times 10^{-10})$

Table 3.6: Truth table of $(\mathsf{x}_1 \wedge \mathsf{x}_2) \wedge (\mathsf{x}_3 \oplus \mathsf{x}_4)$ and the minimised energy of all possible input state of DBN constructed from confidence rules.

3.4 Approximating WFFs and Training DBNs

Given a training set $\mathcal{D} = \{\mathbf{x}^{(n)} | n = 1, .., N\}$ of N samples we can consider this as an incomplete set of all assignments from unknown formula φ. We will no longer be able to create a DNF to build up a DBN to represent the formula. However, we can assume that an assignment is a preferred assignment if it satisfies rules with high confidence values constructed from assignments in $\mathcal{D}$. This is similar as assuming that a sample belongs to a given class if it shares many patterns with the training samples in this class. In other words, suppose that a DBN is constructed from a incomplete set of preferred assignments. A new assignment is more likely to be a preferred assignment if it has low energy. This is because each confidence rule contributes to the total energy with a negative amount, therefore more confidence rules with high confidence values will reduce the total energy. In order to achieve this we can generate confidence rules from $\mathcal{D}$ and adjust the confidence values so that preferred assignments in $\mathcal{D}$ have lower energy, thus relaxing the constraint that the lower RBMs should have a near-zero confidence value, and the top RBM should have a near-infinite confidence value.

This process of approximating confidence values from an incomplete set of preferred assignments can be seen as similar to a layer-wise training of a DBN given a dataset. Indeed, given a training set $\mathcal{D}$, the first layer will be trained to maximise the log-likelihood which assigns a low energy to training samples (preferred assignments) [53]. After that the hidden states are inferred and used as input to train a higher RBM in the same way. Note that this inference in an RBM can be seen as a stochastic step to find states of hidden units which minimise the energy of the RBM given a visible state. Therefore the learning of DBNs can be seen as an approximation of assigning low total energy to training samples. This instigates the question of extracting symbolic knowledge from DBNs using confidence rules. How would confidence rules perform in comparison with the original DBN? This question will be addressed in next chapter.

3.5 Summary

In this chapter we have discussed the relation between propositional logic and deep belief networks. We investigated correspondences between logical inference and minimising energy functions. We have extended the work on Penalty Logic [109] to construct RBMs and then build DBNs from logical formulas given in a hierarchical form. We have introduced confidence rules as a natural representation for DBNs. We argue that approximating a hierarchical set of confidence rules to represent an unknown formula can be done in a similar way as learning a DBN from a training set. In the next chapter we will study how to extract rules from DBNs through the conversion of DBNs into confidence rules.

Chapter 4

Deep Belief Logic Networks

The previous chapter showed how confidence rules are related to RBMs and DBNs. Representing a trained RBM/DBN in propositional logic, however, is difficult because the weight matrices in a trained RBM/DBN can vary considerably and yet represent the same WFF. In this chapter, we propose a method to extract confidence rules from trained RBMs/DBNs. We show that even though the extracted rules may not represent exactly the models, they can be useful at describing the underlying knowledge obtained from data. Furthermore, in some cases, the accuracies of the extracted rules are significantly close to that of the model. We also investigate how the encoding of prior symbolic knowledge onto RBMs/DBNs can help improve performance at unsupervised layer-wise learning.

4.1 Extracting Confidence Rules from RBMs

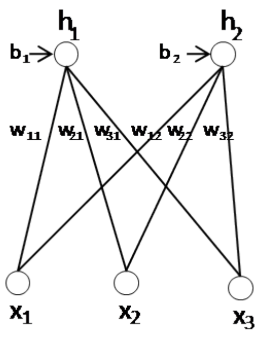
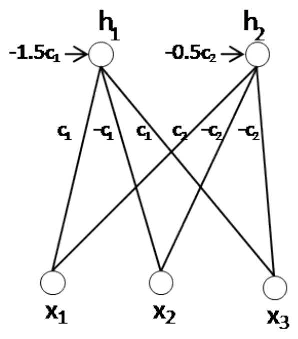
One basic technique to extract rules from trained networks is to consider all possible assignments of input variables. Let us consider a unit h_j in the hidden layer of an RBM from which a confidence rule, given a state of the visible layer, can be extracted as follows:

$$c_j(\mathsf{x}) : \mathsf{h}_j \leftrightarrow \bigwedge_{t \in T_j} \mathsf{x}_t \wedge \bigwedge_{k \in K_j} \neg \mathsf{x}_k$$

where x_t and $\neg\mathrm{x}_k$ denote that the visible units x_t and x_k have values 1 and 0 respectively. The confidence value can be computed, for example, as $c_j(\mathbf{x}) = 1 + \exp(\sum_{i=\{t,k\}} w_{ij}x_i)$ so that the product of all confidence values of the rules that match the assignment $\mathbf{x}$ is proportional to the probability of $\mathbf{x}$ in the RBM. This is because $p(\mathbf{x}) = \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}))}{Z}$ while $\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h})) = \prod_j(1 + \exp(\sum_{i=\{t,k\}} w_{ij}x_i))$. By doing this one can extract confidence rules that perfectly represent the RBM. However, the number of possible states of a layer is exponential on the number of units, which causes the extraction in this way to become intractable for large RBMs.

Since extracting symbolic rules that are equivalent to the original RBMs is difficult, heuristic approaches have been applied to obtain sets of rules that in some cases can perform as effectively as the network models. In Neural-Symbolic Cognitive Agents (NSCA) [106], a sampling method has been proposed to extract temporal logical rules from recurrent temporal RBMs. These rules have been used to evaluate driving skills, and achieved similar performance as driving instructors. Applying NSCA to RBMs, one can extract rules by setting each hidden unit as activated one at a time, and performing downward inference. The rule is constructed similarly to confidence rules in that if the probability of a visible unit being activated is larger than a threshold (for example, 0.5) then this unit will be represented by a positive proposition in the rule; otherwise it will be a negative proposition. The "confidence/confidence level"' of a rule in NSCA is computed as the probability of the hidden unit given the truth-value assignment that satisfies the rule. NSCA then creates propositions containing all the variables. In this thesis, a new extraction method is proposed that seeks to effectively and efficiently extract knowledge from RBMs by converting them into an RBM from which extracting confidence rules is straightforward, as illustrated below (where a trained RBM containing a set of weight in the real numbers is converted into an RBM with weights c_1, $-c_1$, c_2, $-c_2$ only, from which the confidence rules below can be extracted directly). Different from the "confidence/confidence level" in NSCA which is a probability, the confidence value of the confidence rules is a non-negative real number. Furthermore, while rule extraction in NSCA includes the inference of visible layer after activating a hidden node, our rule extraction only considers the values of the weight matrix.

There is, of course, no guarantee that the converted RBM will be equivalent to the original one, but we expect some extracted rules to be useful, as our evaluation

Trained RBM	Converted RBM	Confidence rules
 h1 h2 b1 b2 w11 w21 w31 w12 w22 w32 x1 x2 x3	 h1 h2 -1.5c1 -0.5c2 c1 -c1 c1 c2 -c2 -c2 x1 x2 x3	$c_1 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3$ $c_2 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \neg \mathsf{x}_3$

indicates (e.g. as a compact representation for the RBMs without too much loss of accuracy). In the next section, we will introduce a basic method to perform the above conversion, and after that we will give an example of how useful knowledge can be discovered.

4.1.1 Minimising Euclidean Distance

This section proposes an algorithm to extract confidence rules from a trained RBM. The objective, as mentioned earlier, is to convert the original RBM into an RBM from which confidence rules can be extracted directly without too much loss of accuracy. Each sub-network consisting of a hidden unit, all visible units and their connection weights will be converted to a new sub-network of the simplified RBM. This conversion will be done by calculating the confidence-value c_j and the sign of the new weight s_{ij} from visible unit i and hidden unit j, as illustrated in Figure 4.1.

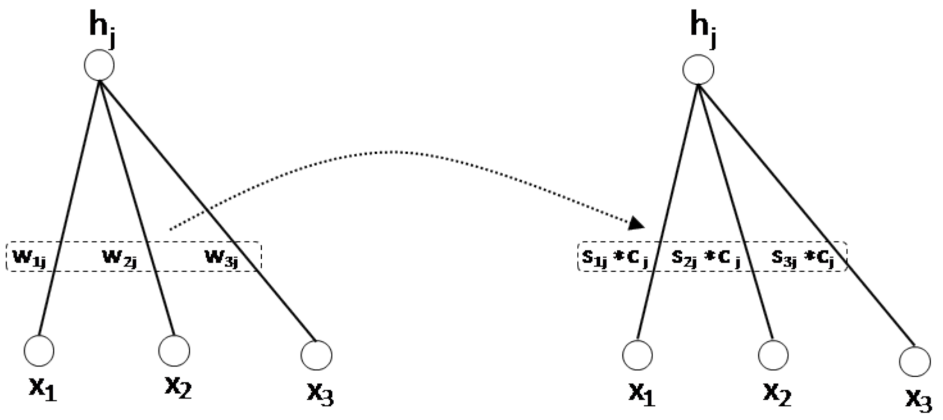


Figure 4.1: Converting a sub-network in original RBM to new sub-network in easy-to-interpret RBM.

Formally, for each hidden unit j of the original RBM we convert the column vector $\mathbf{w}_j$ into a new vector $\mathbf{s}_j$ associated with a confidence value c_j so that the confidence rule below can be extracted:

$$c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{s_{tj}>0} \mathsf{x}_t \wedge \bigwedge_{s_{kj}<0} \neg \mathsf{x}_k \tag{4.1}$$

The idea of the extraction is to identify the positive and negative propositions and also the confidence value c_j that minimises the (squared) Euclidean distance between $\mathbf{w}_j$ and $c * \mathbf{s}_j$, as follows:

$$D_{euclidean} = \sum_{ij} ||w_{ij} - c_j s_{ij}||^2 \tag{4.2}$$

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

$$s_{ij} = \begin{cases} 1 & \text{if } \mathsf{x}_i \text{ appears in rule } j; \\ -1 & \text{if } \neg\mathsf{x}_i \text{ appears in rule } j; \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

The reason why s_{ij} can be 0 is that, differently from NSCA, we do not necessarily require all propositions to appear in the rule. This should allow the extracted rules to highlight some interesting relationships between the variables, as will be illustrated later in a brief discussion about interpretability of rules.

Since Eq. 4.2 is a quadratic function, the *confidence values* we are looking for can be found by setting the derivatives to zeros, as follows.

$$\sum_i 2(w_{ij} - c_j s_{ij}) s_{ij} = 0, \text{ for all } j \tag{4.4}$$

from which we obtain:

$$c_j = \frac{\sum_i w_{ij} s_{ij}}{\sum_i s_{ij}^2} \tag{4.5}$$

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

$$||w_{ij} - c_j s_{ij}||^2 = ||\text{abs}(w_{ij}) - c_j \frac{s_{ij}}{sign(w_{ij})}||^2 = \\ = \begin{cases} (\text{abs}(w_{ij}) + c_j)^2 & \text{if } s_{ij} \neq sign(w_{ij}); \\ (\text{abs}(w_{ij}) - c_j)^2 & \text{if } s_{ij} = sign(w_{ij}); \\ \text{abs}(w_{ij})^2 & \text{if } s_{ij} = 0. \end{cases} \tag{4.6}$$

Here, abs(w_{ij}) and sign(w_{ij}) are functions that return 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 distance will be minimised if $s_{ij} = \text{sign}(w_{ij})$ or $s_{ij} = 0$. In particular, $s_{ij} = 0$ will minimise the distance function if and only if:

$$\begin{aligned} &\text{abs}(w_{ij})^2 \leq (\text{abs}(w_{ij}) - c_j)^2 \\ &c_j \geq 2 \times \text{abs}(w_{ij}) \end{aligned} \tag{4.7}$$

From Eq. 4.5 and Eq. 4.7, we derive the extraction algorithm below.

Algorithm 1 RBM_EXTRACT

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

4.1.2 Interpretability

We mentioned earlier that although equivalence between rules and RBMs is not guaranteed, we hope that the rules can capture interesting knowledge, in that they are more compact, and hopefully more interpretable, than NSCA rules. Although we do not make claims of interpretability, and will evaluate the extraction method w.r.t. the accuracy of the rules in relation to that of the RBMs, the rules being compact is a relevant property of the extraction, as illustrated below.

A confidence rule is different from classical logic. Given a confidence rule, what can we conclude? For example, how do we reason with the rule:

$$0.5 : h_1 \leftrightarrow \mathit{the\_dog\_plays\_in\_the\_garden} \wedge \mathit{the\_sprinkler\_is\_on} \wedge \mathit{the\_dog\_is\_wet}$$

Given that the dog plays in the garden while the sprinkler is on, one can conclude that the dog is wet, satisfying hypothesis h_1. If the sprinkler is off then we cannot conclude anything about whether the dog is wet because the conjunction is $false$ and h_1 will never be satisfied. Similarly, if we observe that the dog is wet and it plays in the garden then we can conclude that the sprinkler is on. However, the difference between *the_sprinkler_is_on* and *the_dog_is_wet* might be that the former is a causal factor (i.e. a non-target proposition) of the latter (i.e. a target proposition). This means that there may exist contradictory non-target propositions in a rule having the same target proposition. For example, suppose we have another rule:

$$0.7 : h_2 \leftrightarrow \mathit{the\_dog\_plays\_in\_the\_garden} \wedge \mathit{it\_rains} \wedge \neg \mathit{the\_sprinkler\_is\_on} \wedge \mathit{the\_dog\_is\_wet}$$

Given that the dog plays in the garden while it rains and we observe that the dog is wet then the first rule concludes that the sprinkler is on while the second rules states that the sprinkler is off. This is where the confidence values come in, indicating, in this example, a preference for the second rule, and therefore that the sprinkler is off.

Given a rule extraction algorithm, one can check whether the accuracy of the extracted rules approaches that of the network on a data set. One can also evaluate rule fidelity to the network where, instead of accuracy w.r.t. ground truths in the dataset, what matters is that the rules mimic the results of the network, whether those are correct or not. Interpretability, however, is more subjective and domain dependent. In a given application domain, if a domain expert can inspect the rules and find new knowledge then the extraction has been justified. In what follows we exemplify the idea, although our evaluation in the next chapters will be based on accuracy and fidelity instead.

Let us start with the XOR example used earlier and then the car evaluation problem [147].

Inference with weighted symbolic rules can be done through a standard weighted MAX-SAT algorithm [112, 50]. Given premises (truth-value assignments to some propositions), the algorithm will search for the the truth-values of the other propositions (called predicted propositions) that maximise the total weight of the rules that are satisfied (i.e. true). In the case of confidence rules, we have seen that satisfying a rule contributes a negative amount to the energy function of a corresponding RBM. Therefore, inference with confidence rules is to find the truth-values of the predicted propositions such that the sum of the confidence values of all satisfied propositions is maximised.

XOR example:

x	y	z
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*false*

Table 4.1: Truth-table of XOR function.

The XOR example shows how confidence rules extracted from an RBM trained on the XOR function look like. The training examples are the preferred assignments of the XOR: $(\mathsf{x} \wedge \mathsf{y}) \leftrightarrow \mathsf{z}$. An RBM with visible units $\{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$ and 10 hidden units was trained to learn the truth-table in Table 4.1 with input value 0 used to denote truth-value *false*, and 1 to denote *true*.

In this example, ten rules exist with antecedents x, y and z, and consequent h_i, $1 \leq i \leq 10$. The rules extracted from the trained RBM are shown below:

A rule such as $1.340 : \mathsf{h}_1 \leftrightarrow \mathsf{x} \wedge \neg\mathsf{y} \wedge \mathsf{z}$ can be interpreted as "if $\mathsf{x} = \mathit{true}$ and $\mathsf{y} = \mathit{false}$ then z should be *true* to satisfy the hypothesis h_1". Given the truth-values of x and y one can predict the truth-value of z using MAX-SAT as discussed earlier. Table 4.3 contains an example.

$1.340 : h_1 \leftrightarrow x \wedge \neg y \wedge z$	$1.677 : h_6 \leftrightarrow \neg x \wedge y \wedge z$
$2.970 : h_2 \leftrightarrow x \wedge \neg y \wedge z$	$2.544 : h_7 \leftrightarrow x \wedge \neg y \wedge z$
$6.165 : h_3 \leftrightarrow \neg x \wedge y \wedge z$	$7.355 : h_8 \leftrightarrow x \wedge y \wedge \neg z$
$0.158 : h_4 \leftrightarrow \neg x \wedge y \wedge \neg z$	$6.540 : h_9 \leftrightarrow \neg x \wedge \neg y \wedge \neg z$
$2.481 : h_5 \leftrightarrow x \wedge \neg y \wedge z$	$4.868 : h_{10} \leftrightarrow \neg x \wedge \neg y \wedge \neg z$

Table 4.2: Rules extracted from RBM trained on XOR function.

x	y	Total confidence if z = *false*	Total confidence if z = *true*	Conclusion
false	false	11.408	0	z = *false*
false	true	0	8.00	z = *true*
true	false	0	9.335	z = *true*
true	true	7.355	0	z = *false*

Table 4.3: Inference of z from the confidence rules extracted from an RBM trained on the XOR truth-table.

Notice that, if either x or y were chosen as target variable, the same procedure above could be applied, without the need for retraining the RBM. Differently from extraction from supervised models [136, 31], here the target does not have to be chosen in advance or the model retrained for each target.

Car Evaluation example:

Let us now exemplify interpretability of confidence rules in the car evaluation dataset [147]. We choose this dataset because it is easy to interpret, and the data consists of all possible preferred assignments. The car evaluation dataset has 6 variables and one label, all categorical as shown in Figure 4.4. The data consists of

Variable name	label	Possible values
Buying price	No	low, medium, high, very high
Maintenance price	No	low, medium, high, very high
Number of doors	No	2, 3, 4, more than 5
Number of person to carry	No	2, 4, more than 4
Size of luggage boot	No	small, medium, big
Safety	No	low, medium, high
Evaluation	Yes	unacceptable, acceptable, good, very good

Table 4.4: Car evaluation data description. In the second ("label") column "Yes/No" indicates whether the variable is the label or not.

1,728 samples covering the non-target variable space. Since the data is categorical while our examples so far have focused on binary data, we proceed as follows:

For the learning, each input variable is represented by a group of units [53]. Only one unit of the group will receive a value of 1 while the other units are 0. For example, a group of units for the "safety" variable can have three possible states [1 0 0], [0 1 0] and [0 0 1] for "low", "medium" and "high", respectively.

For the rule extraction from a group of units for a discrete variable, we take the highest positive weight of a group as the best representative for that group.

We trained RBMs and extracted confidence rules from them. The best training-set accuracy of the rules using MAX-SAT was 80.25% compared with 91.32% from the RBM. As discussed earlier, a loss of accuracy is expected. Nevertheless, the rules may be useful if they provide interpretable knowledge, as discussed below.

Examples of the extracted rules are provided below. More rules can be found in Appendix C:

- $h_1 \leftrightarrow safety\_is\_low \wedge the\_car\_is\_unacceptable$;
  $h_2 \leftrightarrow can\_carry\_2\_people \wedge the\_car\_is\_unacceptable$, indicating that the car is unacceptable, in this case, if safety is low or it can carry 2 people only.
- $h_8 \leftrightarrow buying\_price\_is\_high \wedge maintenance\_price\_is\_high \wedge can\_carry\_4\_people \wedge safety\_is\_high \wedge the\_car\_is\_acceptable$; a car with high buying price and maintenance cost is acceptable if it can carry 4 people and its safety is high.
- $h_{19} \leftrightarrow buying\_price\_is\_low \wedge maintenance\_price\_is\_low \wedge can\_carry\_4\_people \wedge luggage\_boot\_size\_is\_big \wedge safety\_is\_medium \wedge the\_car\_is\_good$; even though safety is medium, a car is good if it has low buying and maintenance costs, can carry four people, and has a big luggage boot.
- $h_{18} \leftrightarrow buying\_price\_is\_low \wedge maintenance\_price\_is\_low \wedge can\_carry\_more\_than\_4\_people \wedge luggage\_boot\_size\_is\_big \wedge safety\_is\_high \wedge the\_car\_is\_very\_good$. This is similar to hypothesis h_{19} but a car is better if safety is high and it can carry more people.

Next, we define precisely how hierarchical reasoning can be carried out using confidence rules.

4.2 Partial Models

In this section, we define precisely how hierarchical inference can be done as motivated in the Introduction. This makes confident rules behave as a discrete version of RBMs, which we call "partial-models".

Many logic programming systems have hierarchical rules in which *intermediate literals* exist [85]. An intermediate literal is a proposition (also called a literal) that appears in the antecedent (or body) of some rule and in the consequent (or head) of some other rule. In confidence rules, a hypothesis h_j can be considered an intermediate literal. If we assume that each assignment of an input variable is also associated with a confidence value then the inference of intermediate literals can be done through combinations of these confidence values. By doing this, a set of confidence rules can be seen as an RBM with a discrete weight matrix, which we call "partial-model", as defined in the next section.

4.2.1 Hierarchical Inference

Seeing confidence rules as logic programs, the rules are organised into hierarchies and inference is performed bottom-up. Let us consider input variables where each variable x_i can receive a real value from 0 to 1. Given a state of visible variables where $x_i = \alpha_i$ we can convert it into a ***belief that*** $\mathsf{x}_i = true$ ***with confidence value*** α_i. Here we make we make a distinction between two different types of confidences, one for the rules and the other for the propositions. For each subset of rules in the hierarchy, the confidence value of each hypothesis (intermediate literal) in this subset can be inferred, given the confidence value of each belief, and then normalised to be used as beliefs in the inference at the next level of the hierarchy. The following definition formalises this idea.

Definition 4.2.1. *Let $R^{(1)}$ be a set of confidence value rules relating a set of beliefs $\mathsf{x}_1, \mathsf{x}_2, ...$ and a set of hypotheses $\mathsf{h}_1^{(1)}, \mathsf{h}_2^{(1)}, ...$; let $R^{(2)}$ be a set of confidence value rules relating hypotheses $\mathsf{h}_1^{(1)}, \mathsf{h}_2^{(1)}, ...$ and new hypotheses $\mathsf{h}_1^{(2)}, \mathsf{h}_2^{(2)}, ...$; let $R^{(3)}$ be a set of confidence value rules relating hypotheses $\mathsf{h}_1^{(2)}, \mathsf{h}_2^{(2)}, ...$ and new hypotheses $\mathsf{h}_1^{(3)}, \mathsf{h}_2^{(3)}, ...$, and so on. We call $R^{(1)}, R^{(2)}, R^{(3)}, ...$ a hierarchical weighted knowledge-base.*

Given an input to any component $R^{(1)}, R^{(2)}, R^{(3)}, ...$ of a hierarchical weighted knowledge-base, local inference can be carried out and results propagated to components immediately above it in the hierarchy. 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 the inference to work with real-valued data types through the application of the following inference rule.

INFERENCE RULE 01: INF_PARTIAL

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 $\alpha_h = c \times (\sum_{t'} \alpha_{t'} + \sum_{m \neq \forall t'}^{m \in T} \alpha_m - \sum_{k'} \alpha_{k'} - \sum_{m \neq \forall k'}^{m \in K} \alpha_m)$
where $\alpha_m = \frac{min+max}{2}$

In the inference rule INF_PARTIAL, T and K are sets of positive and negative literals, respectively. The *confidence value* α_m for any missing beliefs is the average of an upper-bound and a lower-bound on the normalised *confidence values* in the program. The upper-bound (*max*), the lower-bound (*min*), and the normalisation function are defined according to the extraction algorithm. For example, with rules extracted from binary RBMs/DBNs (Algorithm 1) the *min*, *max* values are 0 and 1 respectively and the normalisation function is the sigmoid function. However, if rules are extracted from a top RBM with label (as will be discussed in §4.3.2) we can define different *min*, *max* values and normalisation function to capture the discriminative relation between non-target variables and the target variable.

Example 4.2.1. For a rule $1.5 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3$, given the premises $1 : \mathsf{x}_1$ and $1 : \mathsf{x}_3$ the inference works as follows:

$$\begin{array}{l} 1.5 : \mathsf{h} \leftrightarrow \mathsf{x}_1 \wedge \neg \mathsf{x}_2 \wedge \mathsf{x}_3 \\ 1 : \mathsf{x}_1 \\ 1 : \mathsf{x}_3 \\ \hline 2.25 : \mathsf{h} \text{ with } 2.25 = 1.5 \times ((1+1) + 0 - 0 - 0.5). \end{array}$$

According to INF_PARTIAL, the sum of the confidences of all the premises that match the positive literals in the rule ($\sum_{t'} \alpha_{t'}$), x_1 and x_3, is $1 + 1 = 2$. Since there are no other positive literals in the antecedent of the rule, both x_1 and x_3 are given as premises, then $\sum_{m \neq \forall t'}^{m \in T} \alpha_m = 0$. The sum of the confidences of all the premises that fail to the negative literals ($\sum_{k'} \alpha_{k'}$) is also 0 because premise x_2 is missing. Suppose that the confidence values of the premises in this case is bound between 0 and 1, then the missing premise can be given a neutral confidence 0.5, which is the last sum ($\sum_{m \neq \forall k'}^{m \in K} \alpha_m$) in INF_PARTIAL. This illustrates how the confidence values of premises and the structure of the rules can be used to calculate the confidence of the hypothesis. If we represent the rule in the form of a discrete vector [1.5 −1.5 1.5] and the confidence of all premises (including the missing one) as a vector then the inference can be done through dot product as:

$$[1.5 \ \ -1.5 \ 1.5][1 \ 0.5 \ 1]^{\top}$$

In this process, with normalisation bounded by $[min, max]$, if $\neg\mathsf{x}$ has confidence value α then x must have confidence value $min + max - \alpha$. The following algorithm formalises the inference process.

Algorithm 2 Bottom-up Inference

```
1: Initialise a set of beliefs $B = \{\alpha_i : \mathsf{x}_i\}$, where each belief has a value $\alpha_i$; $\alpha_i$ can be
   seen as the input value of visible unit $v_i$ corresponding to $\mathsf{x}_i$
2: for $l = 1$ to $L$ do
3:     for each rule $j$ in level $l$ do
4:         Infer $c_j : h_j^l$ from rule $j$ and $B$ using INF_PARTIAL
5:         Add $c_j : h_j^l$ to $H$
6:     end for
7:     Normalise $H$ by setting $c_j := f(c_j)$ such that $c_j \in [min, max]$
8:     Re-write hypotheses $H$ as a set of beliefs $B$
9: end for
```

4.2.2 Low-cost Representation

In RBMs, the state of the hidden units given the state of the visible units can be used as latent features which, in many cases, can be used to improve the training of a classifier [111, 79, 78, 73]. In a confidence rule, given the confidences of the beliefs (i.e. antecedents) we can infer the confidences of the hypotheses (i.e. consequent) using INF_PARTIAL. These confidence values can also be used as input to a classifier. We now show that in addition the confidence rules can be seen as a low-cost representation of the RBMs. In this context, the term "low-cost" refers to the efficient use of memory. In this experiment we apply confidence rules in the form of partial models to extract latent features from images and use such features to train a classifier. We compare the performance of the features extracted from the rules with ones extracted from RBMs through the prediction accuracy of the classifier on the test sets.

We performed experiments with the MNIST handwritten digits dataset, TiCC handwritten characters dataset and YALE face dataset. In each dataset, we divide the data into training, validation and test sets. For the MNIST dataset, we use a subset of the training data with $10,000$ samples (MNIST_{10K}), 2000 validation samples, and $10,000$ test samples for a digits recognition task (from 0 to 9). We also use the same test set to test the confidence rules extracted from RBMs trained on the entire training set with $60,000$ samples[1] (MNIST_{60K}). The TiCC dataset consists of $18,189$ training samples, $1,250$ validation samples, and $18,177$ test samples for a person's letter recognition task (from A to Z). We divide the YALE dataset into a training set with 135 samples, thus 9 samples per person, and the test set with 30 samples. We used an SVM with Gaussian kernel as a classifier to measure the performance of the extracted low-cost representation in comparison with the RBMs. Model selection is performed by running a grid-like search (except for the YALE dataset) over the learning rates for the RBMs (between 0.001 and 1), cost (between 0.0001 and 100), and gamma (between 0.0001 and 100)) for the SVM, all on a log-scale. We did not select the number of hidden units in the RBMs, instead we tested RBMs with 500 and 1000 hidden units only, simply to investigate whether the size of the network affects the quality of the extracted rules.

The memory needed by each type of representation, i.e. RBMs and our low-cost representation using the confidence rules, can be defined as follows:

$$\begin{aligned} M_{RBM} &= T \times C_{word} \times I \times J \\ M_{low-cost} &= (2 \times I \times J) + (T \times C_{word} \times J) \end{aligned} \tag{4.8}$$

where I and J are the number of units in visible layer and hidden layer respectively; C_{word} is the number of bits of a computer word in a device; and T is the number of computer words of a real-valued data type. For example, in a 32-bit machine, an RBM with 784 visible units and 500 hidden units will cost $2 \times 32 \times 784 \times 500 = 25,088,000$ bits for a double precision floating point type. In the case of an implementation of confidence rule in a computing device which

[1]Here, we re-use the hyper-parameters from the experiment with $10,000$ training samples

needs 2 bits to represent a literal in the rules then the memory cost should be $(2 \times 784 \times 500) + (2 \times 32 \times 500) = 816,000$ bits. The ratio of memory saved by the rules over the RBM can be measured by:

$$r_{save} = \frac{M_{RBM} - M_{rules}}{M_{RBM}} \times 100\% \tag{4.9}$$

	float	double
r_{save} no pruning	93.622%	96.747%
r_{save} 20% pruning	94.898%	97.398%
r_{save} 40% pruning	96.173%	98.048%
r_{save} 60% pruning	97.449%	98.699%
r_{save} 80% pruning	98.724%	99.349%

Table 4.5: The expected memory saving ratios for an RBM with 784 visible units and 500 hidden units using standard floating point data types in a 32-bit computer; *pruning* refers to the percentage of the removed hidden nodes which correspond to the rules with low confidence values.

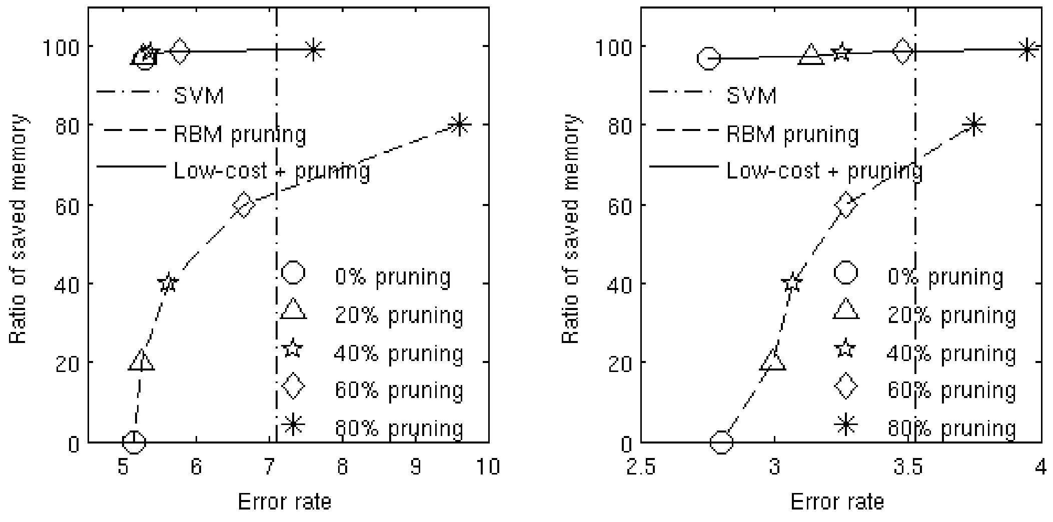
In our experiments, we have trained RBMs using double-precision floating point weight matrices on a 32-bit computer in order to evaluate the performance of the confidence rule in comparison with that of the original RBMs at performing feature extraction. Hence, our purpose is to compare the accuracy and ratio of memory saved of the RBMs and their low-cost representation. We also investigate how accuracy drops as the RBMs are pruned, in comparison with pruning of their extracted rules with respect to the ratio of memory saved. Pruning of x% of a network (RBM or its rules) means that the x% sub-networks corresponding to the rules with the smallest values of c_j, are removed, as done in [138].

	TiCC	MNIST$_{10K}$	MNIST$_{60K}$	YALE face
RBM (J=500)	94.851% ± 0.033	97.198 ± 0.060	98.553% ± 0.031	95.000% ± 2.833
Low-cost	94.711% ± 0.072	97.240 ± 0.089	98.530% ± 0.040	94.333% ± 3.865
RBM (J=1000)	94.928% ± 0.016	97.245% ± 0.031	98.680% ± 0.024	97.000% ± 2.919
Low-cost	94.729% ± 0.070	97.219% ± 0.056	98.562% ± 0.035	96.667% ± 1.757

Table 4.6: Average test set performance of RBMs in comparison with their low-cost representation on four different datasets. The table shows the prediction accuracy of the SVMs trained on the features extracted from the model (RBMs) and the features extracted from the rules (Low-cost).

Table 4.6 contains the accuracies of the RBMs with 500 and 1000 hidden nodes trained on four datasets, and the accuracies of the extracted rules, all on the held-out test sets. We have run each experiment 10 times and report the mean accuracy, along with standard deviation. The results show that the performance of the low-cost confidence rules can be almost identical to that of the RBMs, with high consistency.

Next, we evaluate the effectiveness of the extracted rules in comparison with pruning the RBM. For both the RBM and its extracted rules, one can rank and remove the rules with small confidence value. For the sake of comparison, we prune 20%, 40%, 60% and 80% of both the RBMs and the extracted rules, and evaluate performance. As expected, the average test set error increases rapidly with the pruning. However, results show that more than 98% memory saving can be achieved by the low-cost representation from the extracted rules with the feature extraction still offering a significant improvement on the baseline SVM classification obtained from the input data directly.



(a) TiCC dataset. (b) MNIST dataset.

Figure 4.2: Error rate progression in comparison with memory capacity gains for RBMs and the extracted rules pruned by 0, 20, 40, 60 and 80%.

In order to show the usefulness of the compressed representation in the form of confidence rules at feature extraction, we use the classification accuracy obtained by an SVM on the original input data as baseline. We found that for the MNIST_{60K} and YALE face datasets, the features extracted by either the RBM or the confidence rules produced only a slight improvement on the original data trained using an SVM. In the experiments with the TICC and MNIST_{10K} datasets, however, feature extraction outperformed the SVMs. Therefore, we have chosen the latter two datasets to visualise and evaluate the effect of pruning, as shown in Figure 4.2 for RBMs containing 500 hidden units only.

In Figure 4.2, the SVM line indicates the test set error on the raw input data without using RBMs at all. This line separates the space into an area where the use of an RBM, extracted rules or otherwise, can improve performance (on the left hand side) and an area where feature extraction, whichever the memory capacity gains, is not warranted (on the right hand side). Notice that, in the case of the MNIST dataset, since a 0.2% increase in accuracy is generally accepted as a significant improvement [70], Figure 4.2 shows that approximately 98% of memory capacity gains can be obtained from storing a confidence rule for feature extraction, while preserving a significant improvement over the baseline SVM classification applied to the raw input data.

4.3 Extracting Partial-models from DBNs

Following a layer-wise approach [54, 12], a hierarchy of confidence rules can be built for the extraction of rules from DBNs through the repeated application of Algorithm 1. 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 1 twice. This will be followed by the presentation of the general case algorithm for rule extraction from DBNs. For evaluation we do not use DBNs as feature extractors as in §4.2.2. Instead, we compare the rule inference with probabilistic inference in DBNs using the conditional distribution $p(y = c|\mathbf{x})$ at the top layer, where both label y and input $\mathbf{x}$ are encoded in the visible layer.

4.3.1 An example: DNA promoter problem

The DNA promoter dataset [136] has 106 examples, each consisting of a sequence of 57 nucleotides (either 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 $n_p = A, n_p = T, n_p = G, n_p = C$.

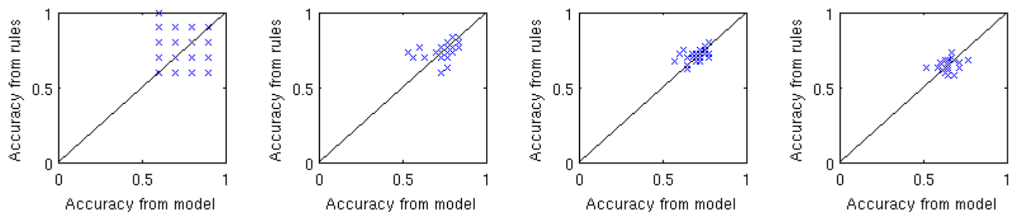
For each variable n_p, a group of 4 visible units, was created in the RBM. Two target units were also added, one for *promoter* and one for $\neg$ *promoter*. Five RBMs were trained using 96 examples, each with 10 examples randomly selected from the original 106 for testing. Only three hidden units were used. After repeating the training and extraction processes, we can find a set of 5 rules with which by applying Inference Rule INF_PARTIAL, all 10 test examples were classified correctly. On average, for the five RBMs, the rules extracted have achieved a test set accuracy of 90% ± 6.9296.

Direct comparisons with other extraction approaches such as MofN [136] and RuleSet [31] would be non-trivial because of the differences in methodology and learning method (supervised vs. 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 10-fold cross-validation, while RuleSet achieved 9 correct classifications out of 10 test set examples on a rule set extracted from a feed-forward neural network trained using back-propagation on the remaining 96 examples.

Exploring the DNA promoter experiment more systematically, let us now evaluate empirically the impact of the performance 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 4 different partitions of training and test data, as shown in Figure 4.4. For each partition, 20 networks were trained using different settings. The graphs then 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 towards the models.

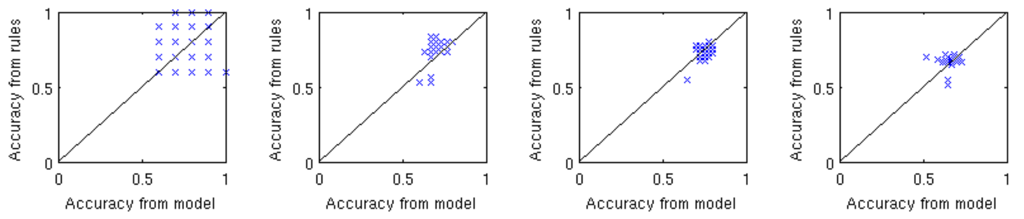
4.3.2 Knowledge Extraction in the Top Layer

We now turn our attention to the special nature of certain nodes in the network, as seen in the case of DNA promoter problem, and notably when the target nodes are



(a) 96/10(train/test). (b) 76/30(train/test). (c) 66/40(train/test). (d) 46/60(train/test).

Figure 4.3: Classification performances of RBMs and the extracted rules on DNA promoter dataset.



(a) 96/10(train/test). (b) 76/30(train/test). (c) 66/40(train/test). (d) 46/60(train/test).

Figure 4.4: Classification performances of DBNs compared with extracted rules on the DNA promoter dataset.

expected to be exclusive (e.g. as part of target layer in the network). In this case, the rules extracted are expected to follow the conditional distribution[2]:

$$p(y = o|\mathbf{x}) \propto \prod_j (1 + e^{\sum_i w_{ij}x_i + u_{oj}}) \tag{4.10}$$

where $\mathbf{U}$ is the weight matrix between the label layer Y and the hidden layer H. Here, y denotes the label (target variable) and $\mathbf{y}$ is its one-hot vector representation. For example, $y = o$ represents class o where $y_o = 1$ and $y_{o' \neq o} = 0$. Applying the logarithm to Eq. 4.10 we have:

$$\log p(y = o, \mathbf{x}) = \sum_j \log(1 + e^{\sum_i w_{ij}x_i} e^{u_{oj}y_o}) \tag{4.11}$$

Algorithm 1 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 $y = o$ expressed as $y_o = 1$, the exponential in the second product in the above equation can be used to normalise the confidence values $\alpha_j^{(1)}$ of $\mathsf{h}_j^{(1)}$, producing:

[2]We ignore the label biases to give equal preference to all classes.

$$\log p(y = o, \mathbf{x}) = \sum_j \log(1 + \alpha_j^{(1)} e^{u_{oj}}) \tag{4.12}$$

Given Eq. 4.12, for each hidden unit and hypothesis $y = o$, one can extract a rule $e^{u_{oj}} : \mathsf{y}_o \leftrightarrow \mathsf{h}_j^{(1)}$, whose confidence value is $e^{u_{oj}}$. By applying INF_PARTIAL and summing up the confidence values for each hypothesis normalised by $f(\alpha) = \log(1 + \alpha)$, one would obtain the same confidence values as produced by Eq. 4.12. Algorithm 3 formalises the resulting rule extraction for a label layer.

Algorithm 3 TOP_RBM_EXTRACT

```
Require: An RBM with visible layer X, hidden layer H, and label layer Y
 1: R = $\emptyset$, T=$\emptyset$
 2: R = RBM_EXTRACT($N_{RBM(X,H)}$) % $N_{RBM(X,H)}$ is the RBM made by layer $X$ and $H$ only
 3: for each hidden unit $j \in$ H and output unit $o \in$ Y do
 4:     Add a rule : $e^{u_{oj}} : \mathsf{y}_o \leftrightarrow \mathsf{h}_j$ to T
 5: end for
 6: return R,T
```

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

Algorithm 4 DBN_EXTRACT

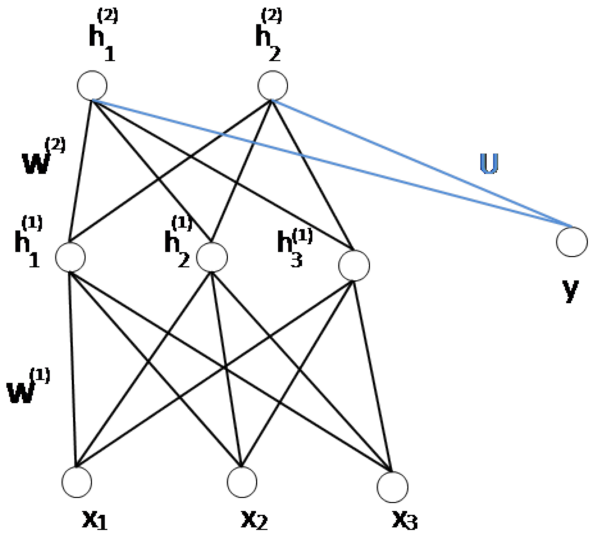
```
Require: A stack of L RBMs: $N_{RBM}^{(1)}, ..., N_{RBM}^{(L)}$; the Boolean flag COMPACT.
 1: Create empty rule set R = $\emptyset$
 2: for $l = 1$ to $L - 1$ do
 3:     R$^{(l)}$ = RBM_EXTRACT($N_{RBM}^{(l)}$)
 4:     Add R$^{(l)}$ to R
 5: end for
 6: if COMPACT then
 7:     R$^{(L)}$ = RBM_EXTRACT($N_{RBM(X,H)}^{(L)}$) % $N_{RBM(X,H)}$ is the RBM made by visible layer $X$ and hidden layer $H$
 8:     T = RBM_EXTRACT($N_{RBM(H,Y)}^{(L)}$) % $N_{RBM(X,H)}$ is the RBM made by hidden $H$ and label layer $Y$
 9: else
10:     R$^{(L)}$,T = TOP_RBM_EXTRACT($N_{RBM}^{(L)}$)
11: end if
12: Add R$^{(L)}$,T to R
```

Example 4.3.1. Suppose we have a trained DBN as in the below DBN with the weight matrices:

$$\mathbf{W}^{(1)} = \begin{pmatrix} 0.1 & -0.2 & 0.3 \\ -0.0001 & 0.2 & -0.3 \\ -0.1 & 0.2 & 0.00001 \end{pmatrix}$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} 0.15 & -0.00001 \\ 0.15 & 0.25 \\ -0.00001 & 0.25 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 0.5 & -0.5 \end{pmatrix}$$



The rules extracted from first RBM are:

$$0.1 : h_1^{(1)} \leftrightarrow x_1 \wedge \neg x_3$$
$$0.2 : h_2^{(1)} \leftrightarrow \neg x_1 \wedge x_2 \wedge x_3$$
$$0.3 : h_3^{(1)} \leftrightarrow x_1 \wedge \neg x_2$$

If COMPACT = *true* the rules extracted from the top RBM are:

$$0.15 : h_1^{(2)} \leftrightarrow h_1^{(1)} \wedge h_2^{(1)}$$
$$0.25 : h_2^{(2)} \leftrightarrow h_2^{(1)} \wedge h_3^{(1)}$$
$$0.5 : y \leftrightarrow h_1^{(2)} \wedge \neg h_2^{(2)}$$

otherwise they will be:

$$0.15 : \mathsf{h}_1^{(2)} \leftrightarrow \mathsf{h}_1^{(1)} \wedge \mathsf{h}_2^{(1)}$$
$$0.25 : \mathsf{h}_2^{(2)} \leftrightarrow \mathsf{h}_2^{(1)} \wedge \mathsf{h}_3^{(1)}$$
$$e^{0.5} : \mathsf{y} \leftrightarrow \mathsf{h}_1^{(2)}$$
$$e^{-0.5} : \mathsf{y} \leftrightarrow \mathsf{h}_2^{(2)}$$

4.3.3 Performance Loss in Complex Domains

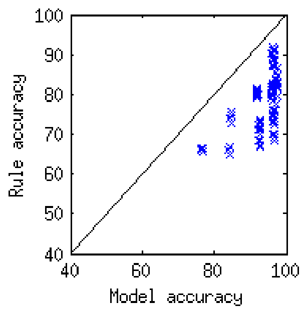
We now test the general method of rule extraction from DBNs (Algorithm 4) on a harder problem, namely the MNIST handwritten digit recognition dataset. This is a difficult problem for rule extraction because the inputs are the values of the pixels in the images to be classified into 10 classes. The rules 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.

In what follows, we report the results using *COMPACT=False* (c.f. Algorithm 4). In the image domain, we found that performance loss is larger when *COMPACT=True*. We attribute performance loss in the case of the MNIST dataset to the fact that the input data is not binary, showing more variance than the DNA data evaluated earlier. As a result, in the case of a deep network, performance loss may be compounded when inference is applied sequentially through the rule hierarchy (i.e. without sampling). In what follows, we evaluate performance loss in more detail.

In Section 4.2.2, 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 layer-wise approach would result in each RBM in the hierarchy being trained and rules extracted before the next RBM can be trained. In order to evaluate performance 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 different learning rates, momentums, and cost in a 2 hidden layer DBN: 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 [144], and 10,000 test examples. Figure 4.5 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 performance loss here than in the case of the DNA data, with an average performance loss of 15.30% ±5.92 in relation to the DBNs.

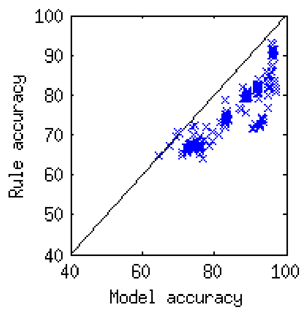
The DBNs were trained using learning rate decay and early stopping based on their performance on the validation set (whenever the validation set error increased, a lower learning rate would be used for network training). The same can be done using rule sets extracted from the network, as follows: rules 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. Figure 4.6 shows a comparison between the test-set accuracy of the DBNs (model accuracy), now using rule-based early stopping, and the test-set accuracy of the extracted rules. Now, an average performance loss of 9.25% ±4.20 is achieved in relation to the DBNs. In comparison with Figure 4.5, 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 performance loss). Given the complexity of image domains when it comes to rule extraction, we interpret the results shown in Figure 4.6 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 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 remainder of this chapter.



(a) 2-hidden layer DBN

Figure 4.5: Comparison between the test-set accuracies of DBNs (model accuracy) and extracted rules.



(a) 2-hidden layer DBN

Figure 4.6: Comparison between the test-set accuracies of DBNs (model accuracy) and extracted rules using rule-based early stopping.

4.4 Deep Neural-Symbolic Integration Systems

Having seen how symbolic knowledge can be extracted from DBNs, we now investigate the inverse problem of inserting 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 [137, 8]. 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.

4.4.1 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 datasets 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 §4.1 and §4.3, 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.

Example 4.4.1 and Figure 4.7 illustrate the main idea behind the encoding algorithm to follow using a simple set of rules. Figure 4.7 also illustrates how the DBN can be extended to account for learning from data and background knowledge, which is discussed in the sequel.

Example 4.4.1. (Encoding knowledge) Given a hierarchical set of rules $K^s = \{K^{(1)}, K^{(2)}, K^{(3)}\}$, where:

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

For a dataset with variables $\{x_1, x_2, x_3, x_4, x_5, t_1, t_2\}$, consider 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. Figure 4.7 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 only support 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.

Algorithm 5 Rule Encoding Algorithm

```
Require: a hierarchical weighted knowledge-base $K$
 1: for $l = 1$ to $L$ do
 2:     Initialise an empty RBM $N^{(l)}$;
 3:     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;
 4:         Add a unit $j$ to hidden layer $l$;
 5:         Set the value of the connection weight $w_{tj}^{l}$ from node $\mathsf{h}_t^{(l-1)}$ to node $j$ to $c_j$;
 6:         Set the value of the connection weight $w_{kj}^{l}$ from node $\mathsf{h}_k^{(l-1)}$ to node $j$ to $-c_j$;
 7:     end for
 8:     if $l > 1$ then
 9:         Stack $N^{(l)}$ on top of $N^{(l-1)}$;
10:     end if
11: end for
```

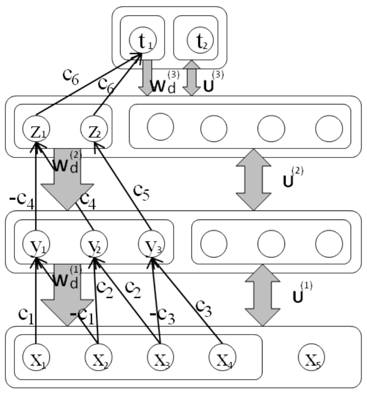


Figure 4.7: DBN obtained from hierarchical rule set K from Example 4.4.1.

4.4.2 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:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_j h_j c_j \sum_i s_{ij} x_i - \sum_{i,k} x_i u_{ik} h_k - \sum_i a_i x_i - \sum_k b_k h_k \tag{4.13}$$

where, x and 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 (c.f. Algorithm 5), and $u_{ij} \in \mathbf{U}$ is the value of the weights of the extra hidden units (see Figure 4.7).

The encoded knowledge will be used to guide learning within the neural-symbolic RBMs by maximising the log-likelihood of the parameters given the data and background knowledge. Since the connections in an RBM are bi-directional, while the background knowledge only supports 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* ($\mathbf{W}_u$), and random values were assigned to the *down-weights* ($\mathbf{W}_d$). The learning algorithm below will, therefore, adapt the parameters that consist of additional connection weights $\mathbf{U}$ and the down-weights $\mathbf{W}_d$ given the confidence values.

We use Contrastive Divergence [52] to train the networks. The log-likelihood function is given by:

$$\mathcal{L}_{lRBM} = \sum_{\mathbf{x} \in \mathcal{D}} P(\mathbf{x} | \theta = \{\mathbf{U}, \mathbf{W_d}, \mathbf{c}\}; \mathcal{K}). \tag{4.14}$$

We call the learning algorithm below *learning with guidance* because prior knowledge is used to partially fix some upward connections in the network; all other connections, both downward and bi-directional, are allowed to change using standard Contrastive Divergence.

Algorithm 6 Learning with Guidance

```
Require: A set of rules $K_l^{(s)}$; input data, MAX_ITER
1: Select a number of rules in $K_l^{(s)}$ with the highest confidence values
2: Encode $K_l^{(s)}$ in hidden units $H^{(s)}$ with up-weights $\mathbf{W}_u^{(l)}$ and random down-weights $\mathbf{W}_d^{(l)}$
3: Add extra hidden units $H^{(t)}$ with weights $\mathbf{U}^{(l)}$
4: for $i = 1$ to MAX_ITER do
       % positive stage: assign the input to visible layer X
5:     $\mathbf{X}_{pos} := input;$
6:     $\mathbf{H}_{pos} := P(H|\mathbf{X}_{pos});$        $\hat{\mathbf{H}}_{pos} \sim P(H|\mathbf{X}_{pos});$
7:     $\mathbf{X}_{neg} := P(X|\hat{\mathbf{H}}_{pos});$        $\hat{\mathbf{X}}_{neg} \sim P(X|\hat{\mathbf{H}}_{pos});$
8: % negative stage
9:     $\mathbf{H}_{neg} = P(H|\hat{\mathbf{X}}_{neg});$
10:    $\mathbf{W}_d^{(l)} = \mathbf{W}_d^{(l)} + \eta(\langle \mathbf{X}_{pos}^\top \mathbf{H}_{pos}^{(s)} - \hat{\mathbf{X}}_{neg}^\top \mathbf{H}_{neg}^{(s)} \rangle)$
11:    $\mathbf{U}^{(l)} = \mathbf{U}^{(l)} + \eta(\langle \mathbf{X}_{pos}^\top \mathbf{H}_{pos}^{(t)} - \hat{\mathbf{X}}_{neg}^\top \mathbf{H}_{neg}^{(t)} \rangle)$
12: end for
13: % MAX_ITER is the number of training epoch which is sellected empirically, i.e. using the validation set.
```

4.4.3 Experiments

Experiments on DNA Promoter dataset

In this experiment, we use the domain theory provided with the DNA promoter dataset[3] to set up and train a DBN. The data has been described in §4.3.1. As before, we use variable n_p to denote a nucleotide at position p such that e.g. a_p for $n_p = A$ ("the nucleotide at position p is type A"). Hence, background 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 a confidence value c. The prior rules provided by domain experts are shown in Table 4.7. The knowledge states that promoters should be able to make contact and have a valid conformation. There are two regions to make contact : minus_{10} and minus_{35}. The contact regions and conformation are created by group of nucleotides. For example, $\mathsf{minus}_{10} \leftrightarrow \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{t}_{-7}$ indicates that the contact region type minus_{10} can be created by nucleotides T, A, T in positions $-12, -11, -7$ respectively.

We consider minus_{10} and minus_{35} and $\mathsf{conformation}$ as intermediate literals and

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

L-1{	minus_{35}	$\leftrightarrow \mathsf{c}_{-37} \wedge \mathsf{t}_{-36} \wedge t_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{a}_{-33} \wedge \mathsf{c}_{-32}$
	minus_{35}	$\leftrightarrow \mathsf{t}_{-36} \wedge \mathsf{t}_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{c}_{-32} \wedge \mathsf{a}_{-31}$
	minus_{35}	$\leftrightarrow \mathsf{t}_{-36} \wedge \mathsf{t}_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{a}_{-33} \wedge \mathsf{c}_{-32} \wedge \mathsf{a}_{-31}$
	minus_{35}	$\leftrightarrow \mathsf{t}_{-36} \wedge \mathsf{t}_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{a}_{-33} \wedge \mathsf{c}_{-32}$
	minus_{10}	$\leftrightarrow \mathsf{t}_{-14} \wedge \mathsf{a}_{-13} \wedge \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{a}_{-10} \wedge \mathsf{t}_{-9}$
	minus_{10}	$\leftrightarrow \mathsf{t}_{-13} \wedge \mathsf{a}_{-12} \wedge \mathsf{a}_{-10} \wedge \mathsf{t}_{-8}$
	minus_{10}	$\leftrightarrow \mathsf{t}_{-13} \wedge \mathsf{a}_{-12} \wedge \mathsf{t}_{-11} \wedge \mathsf{a}_{-10} \wedge \mathsf{a}_{-9} \wedge \mathsf{t}_{-8}$
	minus_{10}	$\leftrightarrow \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{t}_{-7}$
	conformation	$\leftrightarrow \mathsf{c}_{-47} \wedge \mathsf{a}_{-46} \wedge \mathsf{a}_{-45} \wedge \mathsf{t}_{-43} \wedge \mathsf{t}_{-42} \wedge \mathsf{a}_{-40} \wedge \mathsf{c}_{-39} \wedge \mathsf{g}_{-22} \wedge \mathsf{t}_{-18} \wedge \mathsf{c}_{-16} \wedge \mathsf{g}_{-8}$ $\wedge \mathsf{c}_{-7} \wedge \mathsf{g}_{-6} \wedge \mathsf{c}_{-5} \wedge \mathsf{c}_{-4} \wedge \mathsf{c}_{-2} \wedge \mathsf{c}_{-1}$
	conformation	$\leftrightarrow \mathsf{a}_{-45} \wedge \mathsf{a}_{-44} \wedge \mathsf{a}_{-41}$
	conformation	$\leftrightarrow \mathsf{a}_{-49} \wedge \mathsf{t}_{-44} \wedge \mathsf{t}_{-27} \wedge \mathsf{a}_{-22} \wedge \mathsf{t}_{-18} \wedge \mathsf{t}_{-16} \wedge \mathsf{g}_{-15} \wedge \mathsf{a}_{-1}$
	conformation	$\leftrightarrow \mathsf{a}_{-45} \wedge \mathsf{a}_{-41} \wedge \mathsf{t}_{-28} \wedge \mathsf{t}_{-27} \wedge \mathsf{t}_{-23} \wedge \mathsf{a}_{-21} \wedge \mathsf{a}_{-20} \wedge \mathsf{t}_{-17} \wedge \mathsf{t}_{-15} \wedge \mathsf{t}_{-4}\}$
L-2{	contact	$\leftrightarrow \mathsf{minus}_{35} \wedge \mathsf{minus}_{10}\ \}$
L-3{	promoter	$\leftrightarrow$ contact $\wedge$ conformation}

Table 4.7: Hierarchy of rules from background theory in the DNA dataset; 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.

convert two rules

$$c : \mathsf{promoter} \leftrightarrow \mathsf{contact} \wedge \mathsf{conformation}$$

$$c : \mathsf{contact} \leftrightarrow \mathsf{minus}_{35} \wedge \mathsf{minus}_{10}$$

into

$$c : \mathsf{h}^{(2)} \leftrightarrow \mathsf{minus}_{35} \wedge \mathsf{minus}_{10} \wedge \mathsf{conformation}$$

$$c : \mathsf{promoter} \leftrightarrow \mathsf{h}^{(2)}$$

We use Algorithm 5 to encode the background theory into a two layer DBN, and use Algorithm 6 to greedily train each layer at a time as in standard DBNs. Figure 4.8 shows the model being constructed, with five rules:

$$c : \mathsf{minus}_{10} \leftrightarrow \mathsf{t}_{-12} \wedge \mathsf{a}_{-11} \wedge \mathsf{t}_{-7}$$

$$c : \mathsf{minus}_{35} \leftrightarrow \mathsf{t}_{-36} \wedge \mathsf{t}_{-35} \wedge \mathsf{g}_{-34} \wedge \mathsf{a}_{-33} \wedge \mathsf{c}_{-32}$$

$$c : \mathsf{conformation} \leftrightarrow \mathsf{a}_{-45} \wedge \mathsf{a}_{-44} \wedge \mathsf{a}_{-41}$$

$$c : \mathsf{h}^{(2)} \leftrightarrow \mathsf{minus}_{35} \wedge \mathsf{minus}_{10} \wedge \mathsf{conformation}$$

$$c : \mathsf{promoter} \leftrightarrow \mathsf{h}^{(2)}$$

having been encoded.

For evaluation, we partition the data into a training, validation and test sets. In

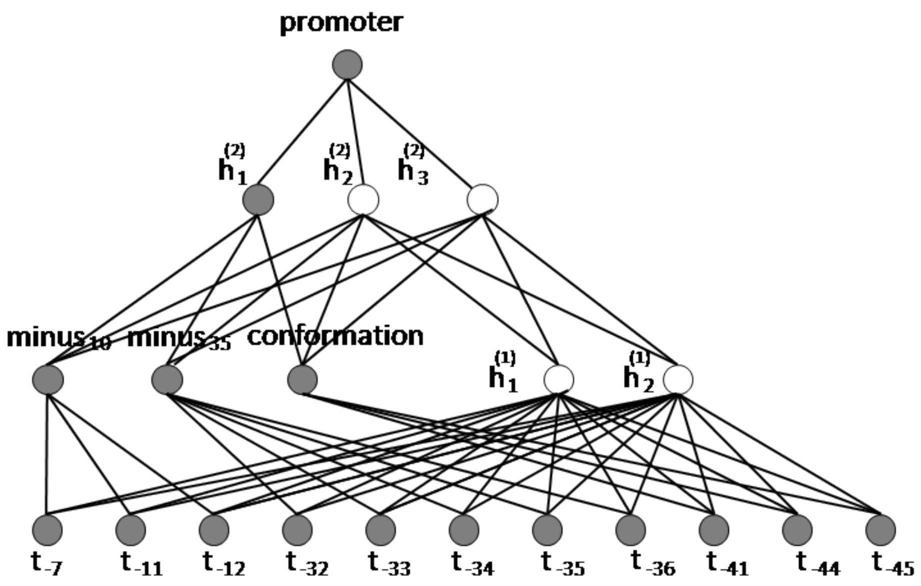


Figure 4.8: A deep neural-symbolic integration with 5 rules have been encoded.

order to investigate how background knowledge influences learning on 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 hyper-parameters is large compared to the small validation and test sets. Hence, the grid search tends to produce a DBN that over-fits the validation set. When background knowledge is encoded into the networks, over-fitting is avoided through the use of the rule guidance learning algorithm, which seems to have led to a more general model.

In order to make a better comparison between the standard DBN and the encoded DBN, we include in Figure 4.9 the best accuracy achieved on the test set using the standard DBN without using model selection. The figure 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 seems to have been able to learn from data the knowledge that had been provided as background theory, and therefore the improvement with prior knowledge

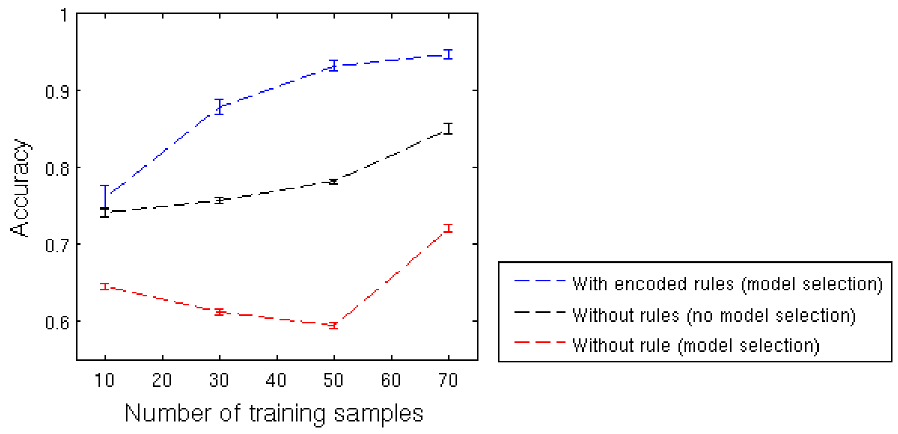


Figure 4.9: Test set classification performance of 2-layer standard DBN (red and black lines, with and without model selection), and 2-layer DBN encoded with prior knowledge from the DNA promoter background theory using training sets with 10, 30, 50 and 70 samples, and 20 test samples.

is smaller for larger training sets.

Experiments on MNIST dataset

In order to evaluate the influence of background knowledge on the MNIST dataset (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 dataset, there were 20,000 examples for training, 10,000 examples for validation, and 10,000 examples for testing. We have selected 1,000 examples randomly from the training set for training a 2-layer DBN from which rules were then extracted. Such rules were encoded into a new DBN, following the procedure above. 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

Figure 4.10 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 1,000 examples, the DBN set-up 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

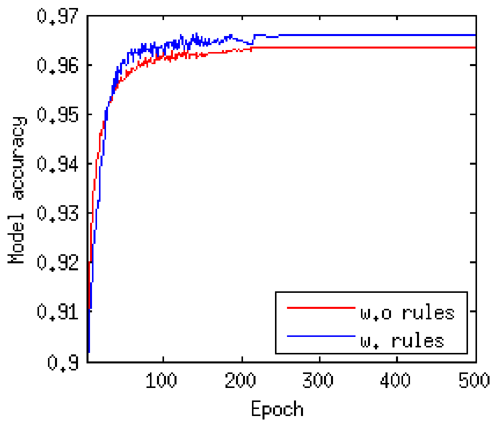


Figure 4.10: Performance of modular DBNs with and without encoded rules.

1,000 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.

4.5 Summary

This chapter 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.

In order to support hierarchical inference in complex domain where the input data is real-valued we extended confidence rules to partial-models. This logical language was designed to support hierarchical reasoning, and was shown to be an adequate representation for the modular training and symbolic representation of deep networks. Knowledge represented by partial-models can be inserted or extracted from DBNs. It is shown that in single-layer DBNs, also known as restricted Boltzmann machines, confidence logic offers a low-cost representation for the RBMs. Yet, the modular training of networks as part of a cycle of knowledge insertion, learning and extraction can produce an improvement in performance. Knowledge encoding into DBNs in the form of confidence logic rules has been shown to be useful, leading to an improvement in performance following a layer-wise training. The results from this work suggest that there is promise in the building of a hierarchical reasoning system capable of integrating symbolic and sub-symbolic capabilities.

Chapter 5

Using Confidence Values for Representation Ranking

Chapter 4 studied knowledge extraction and encoding using deep belief networks. A neural-symbolic model has been proposed to show the effectiveness of using symbolic knowledge to support layer-wise unsupervised learning in a stack of RBMs. However, obtaining symbolic background knowledge from a complex domain can be challenging, requiring considerable manual effort from domain experts. An alternative is to learn and extract knowledge in a (source) domain to transfer and encode it in another (target) domain. In this chapter we discuss the use of confidence values for representation ranking and subsequently transfer learning. First, an efficient method is proposed to compute confidence values. These confidence values will be used to rank the representations in an RBM by decomposing each representation into complete-models. Finally, a novel transfer learning approach is introduced to allow reuse of extracted complete-models.

5.1 Transfer Learning

Normally, in representation learning features are considered to be nonlinear transformations of the original data, which is different from linear techniques such as Principal Component Analysis (PCA) or Singular Vector Decomposition (SVD). An advantage of the component analysis approach is that each component represents a different level of variation in the data, and therefore it can be explained using the corresponding eigenvalue. In contrast, in (nonlinear) representation learning, finding an explanation for the features learned in an unsupervised way becomes more difficult. In Chapter 4, confidence values have been shown useful at describing partially the representations learned, helping identify relevant rules.

In this chapter we investigate that, in the general case, different ways of calculating confidence values are possible. This takes inspiration from the work in logic on labelled deductive systems, where logical sentences are labelled and the algebra used for the labelling can vary, producing different logical systems [41].

The confidence values will be different from the ones in Algorithm 1 RBM_EXTRACT, and will be shown useful at ranking the feature representations, such that if the higher-ranking feature detectors are kept, the network does not lose accuracy abruptly. The rule representation used to allow this are what we call complete-models.

This ranking approach is efficient and independent from the task of the model, different from the mutual information approach [13]. We choose RBMs because the exact training is intractable, and approximate algorithms which have been shown successful such as *Contrastive Divergence* (CD) [52] should be sensitive to noise, and therefore learn high confidence value as well as low confidence value representations. As a result, this efficient confidence value extraction can be useful for network pruning. In what follows, we will show that the use of confidence values is also important for knowledge integration and transfer learning when prior knowledge is not provided.

In Chapter 4, the integration of symbolic knowledge has been studied. However, in many circumstances, symbolic knowledge is difficult to obtain, especially in complex domains such as vision and audio. One may use extraction algorithms to obtain symbolic knowledge from a domain and then use it as background knowledge to support further learning when more examples become available. However, since rules are extracted from a trained network, why would using the rules be better than re-using the network itself? In transfer learning, the idea is to transfer knowledge from another domain to improve learning in a domain of interest.

We will show that the efficient extraction of confidence values for complete models is relevant for representation transfer learning where the learning of new knowledge in a target domain is dependent on the amount of the transferred knowledge from a source domain. In particular, we develop a novel guidance approach to self-taught transfer learning [110] using representation ranking in RBMs, named *Guided Self-taught learning* (GSTL). First, an RBM is trained on a source domain and then its complete-models are transferred to learn additional representations on a target domain. However, transferring a large number of complete-models seems to restrict the learning of new knowledge in the target RBM. Therefore transferring of high-ranking features only from a source to a target domain can reduce the size of the RBM in the target domain, while preserving most of the accuracy achieved from the source domain.

This approach is similar to self-taught learning [110] in that it re-uses (parts of) the trained model, and not the (source domain) data, for the learning in the target domain. More precisely, given a network $N^{\mathcal{S}}$, which has been trained in domain $\mathcal{S}$, we extract knowledge $\theta^{\mathcal{S}}$ from $N^{\mathcal{S}}$ and encode it in a new network $N^{\mathcal{T}}$ to be trained in an analogous domain $\mathcal{T}$. An advantage of self-taught learning is that the representation learned can be re-used for transferring to many different domains. However, it does not take into account the representation which can be learned from target data. The proposed GSTL approach instead trains a target RBM such that the transferred knowledge $\theta^{\mathcal{S}}$ will guide the learning of new knowledge from data in a target domain.

5.2 Feature Selection By Ranking Confidence Values

Feature selection has been studied for many years (c.f. [47] for a survey). The main focus has been the selection of useful attributes from a dataset for a specific task, e.g. classification. Differently from that, representation ranking is more about extraction and selection of knowledge which can capture the most accuracy from a model, as formally defined in **Chapter 4**. Since complete-models capture exactly the representations, the confidence values can be used as scores to rank the representations based on the relations between complete-models and partial-models. In this chapter, instead of using the iterative approach as in Algorithm 1, a more efficient method is introduced.

In what follows, we present a method for selecting representations from an RBM by ranking its feature detectors. Based on §4.1.1 we define a ranking function using the confidence values extracted from RBMs. The difference here is that vector $\mathbf{s}_j$ is a vector of the same size as $\mathbf{w}_j$, and $\mathbf{s}_j \in [-1, 1]^I$, where I is the number of visible units. By omitting the zeros in the vector $\mathbf{s}_j$ the computation of confidence values will be more efficient because it does not need to remove the small weights in $\mathbf{w}_j$, as shown below.

Similarly to equation Eq. 4.2, confidence value c_j can be found by setting the derivatives to zero, such that $c_j = \frac{\sum_i w_{ij} s_{ij}}{\sum_i s_{ij}^2}$. If abs($x$) and sign($x$) are also defined as two functions returning the absolute value of x and the sign (-1 or 1) of x, respectively, then we can see that:

$$\begin{aligned} \|w_{ij} - c_j s_{ij}\|^2 &= \|\text{abs}(w_{ij}) - c_j \frac{s_{ij}}{\text{sign}(w_{ij})}\|^2 \\ &\geq \|\text{abs}(w_{ij}) - c_j\|^2 \end{aligned} \tag{5.1}$$

holds if and only if $s_{ij} = \text{sign}(w_{ij})$, which will also minimise $D_{euclidean}$. Applying this to Eq. 4.5, we obtain:

$$c_j = \frac{\sum_i \text{abs}(w_{ij})}{I} \tag{5.2}$$

It is interesting that the confidence value of a complete-model obtained from Eq. 5.2 is the average strength of all weights in the sub-network represented by that complete-model. This means that the value can be computed efficiently for very large networks. We may notice that using c_j instead of weights results in a compression of the network and that the network's dimensionality can be reduced by ordering and pruning feature detectors (complete-models) with low confidence values. Hence, we are interested in investigating whether the detectors with higher confidence values are useful for transfer learning. In what follows, we use again the XOR example to illustrate that low-scoring detectors are less important than high-scoring ones.

5.2.1 XOR Example revisited

We have trained an RBM with 10 hidden nodes to model the XOR function from its truth-table such that $\mathsf{z} = \mathsf{x}$ XOR y. The *true*, *false* logical values are represented by integers $1, 0$, respectively. After training the network, a score for each sub-network can be obtained. The score allows us to replace each real-value weight by its sign, and interpret those signs logically where a negative sign $(-)$ represents logical negation $(\neg)$, as exemplified in Table 5.1.

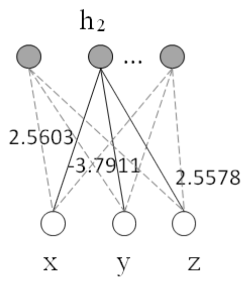
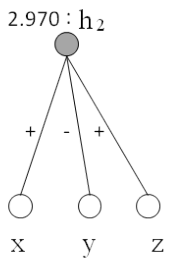
Network	Sub-network	Symbolic representation
		$2.970 : h_2 \sim \{+x, -y, +z\}$ can be interpreted as $\mathsf{z} = \mathsf{x} \wedge \neg \mathsf{y}$ with score 2.970

Table 5.1: RBM trained on XOR function and one of its sub-networks with score value and logical interpretation.

Table 5.2 shows all the sub-networks with associated scores. As one may recognise, each sub-network represents a logical rule learned from the RBM. However, not all of the rules are correct w.r.t. the XOR function. In particular, the sub-network scored 0.158 encodes a rule $\neg\mathsf{z} = \neg\mathsf{x} \wedge \mathsf{y}$, which is inconsistent with $\mathsf{z} = \mathsf{x}$ XOR y. By ranking the sub-networks according to their scores, this can be identified: high-scored sub-networks are consistent with the data, and low-scored ones are not. We have repeated the training several times with different numbers of hidden units, obtaining similar intuitive results. We also call the above scores *confidence values* (for complete models) based on its similarity with the *confidence* of [109, 106, 138] and *knowledge weight* of symbolic knowledge representation [113].

5.2.2 Complete-models

In **Chapter 1** we introduced the concept of complete-models in that they should capture the same rule structure of partial-models while being in addition associated with a set of confidence vectors to represent all the network weights.

Score	Sub-network	Logical Representation
1.340	$h_1 \sim \{+x, -y, +z\}$	$\mathsf{z} = \mathsf{x} \wedge \neg \mathsf{y}$
2.970	$h_2 \sim \{+x, -y, +z\}$	$\mathsf{z} = \mathsf{x} \wedge \neg \mathsf{y}$
6.165	$h_3 \sim \{-x, +y, +z\}$	$\mathsf{z} = \neg \mathsf{x} \wedge \mathsf{y}$
0.158	$\mathbf{h_4} \sim \{-\mathbf{x}, \mathbf{y}, -\mathbf{z}\}$	$\neg \mathbf{z} = \neg \mathbf{x} \wedge \mathbf{y}$
2.481	$h_5 \sim \{+x, -y, +z\}$	$\mathsf{z} = \mathsf{x} \wedge \neg \mathsf{y}$
1.677	$h_6 \sim \{-x, +y, +z\}$	$\mathsf{z} = \neg \mathsf{x} \wedge \mathsf{y}$
2.544	$h_7 \sim \{+x, -y, +z\}$	$\mathsf{z} = \mathsf{x} \wedge \neg \mathsf{y}$
7.355	$h_8 \sim \{+x, +y, -z\}$	$\neg \mathsf{z} = \mathsf{x} \wedge \mathsf{y}$
6.540	$h_9 \sim \{-x, -y, -z\}$	$\neg \mathsf{z} = \neg \mathsf{x} \wedge \neg \mathsf{y}$
4.868	$h_{10} \sim \{-x, -y, -z$	$\neg \mathsf{z} = \neg \mathsf{x} \wedge \neg \mathsf{y}$

Table 5.2: Sub-networks and scores from RBM with 10 hidden units trained on XOR truth-table.

For each hidden unit in an RBM we can extract a complete-model in the form $c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{t,w_{tj}>0}(\frac{w_{tj}}{c_j} : \mathsf{x}_t) \wedge \bigwedge_{k,w_{kj}<0}(\frac{-w_{kj}}{c_j} : \neg \mathsf{x}_k)$, where c_j is the confidence value from Eq. 5.2, representing the exact structure of the RBM. The partial-model of the complete-model

$$c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{t,w_{tj}>0} (\frac{w_{tj}}{c_j} : \mathsf{x}_t) \wedge \bigwedge_{k,w_{kj}<0} (\frac{-w_{kj}}{c_j} : \neg \mathsf{x}_k)(5.3)$$

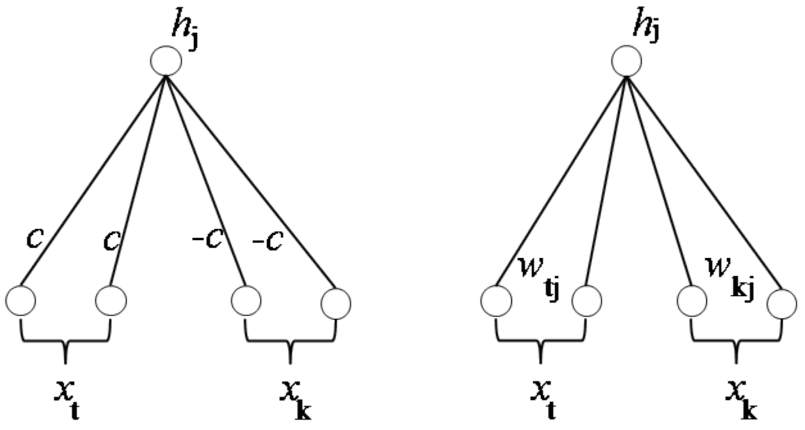
is

$$c_j : \mathsf{h}_j \leftrightarrow \bigwedge_{t,w_{tj}>0} \mathsf{x}_t \wedge \bigwedge_{k,w_{kj}<0} \neg \mathsf{x}_k(5.4)$$

from which we can reconstruct a sub-network as shown in Figure 5.1a

The confidence value of each proposition indicates how strongly a visible unit correlates with the hidden unit. Therefore a connection weight (w_{tj} or w_{kj}) between a visible unit and the hidden unit can be constructed by multiplying c_j or $-c_j$ with $\frac{w_{tj}}{c_j}$ or $\frac{-w_{tk}}{c_j}$ respectively, which results in the sub-network of an RBM, as shown in Figure 5.1b. Applying this to all complete-models we can reconstruct the exact RBM.

It can be seen that complete-models are obtained by decomposing an RBM and keeping the values of the weights in the confidence values. Hence, complete-models represent exactly the feature detectors of an RBM. The advantage of complete-models is we can use their confidence values as scores for ranking the representations.

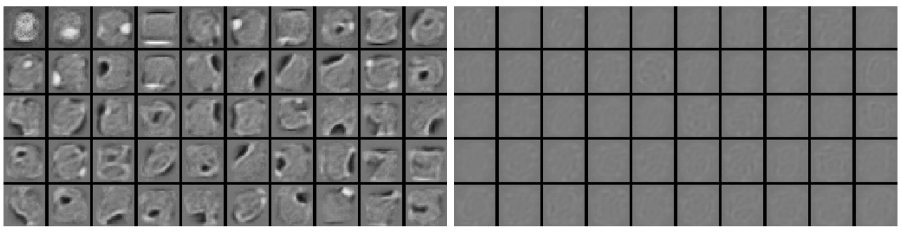


(a) Partial model. (b) Complete model.

Figure 5.1: Sub-networks for partial model in Eq. 5.4 and complete model in Eq. 5.3, respectively. As we can see, the weights in the sub-network representing a partial model have the discrete values $\{c, 0, -c\}$, while the weights of the sub-network representing a complete model have real values, similar as the weights of RBMs.

5.2.3 Representation Ranking

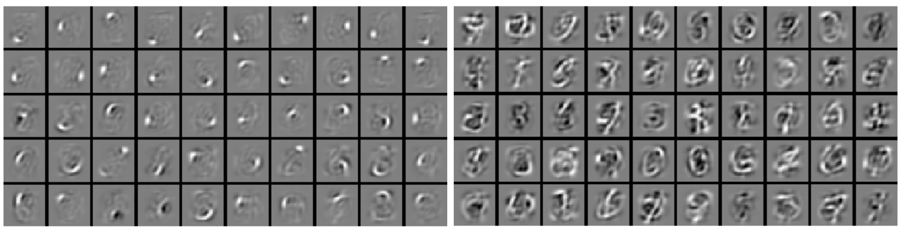
We have trained an RBM with 500 hidden nodes on 20,000 samples from the MNIST dataset in order to visualise the feature detectors of the 50 highest scoring sub-networks and the 50 lowest scoring sub-networks (each takes 10% of the network's capacity). The visualisation is performed by normalising each feature detector (a column vector of the weight matrix) to between 0 and 1, and reshaping the vector to a matrix presenting an image of size 28×28. Figure 5.2 shows the result of using a standard RBM, and Figure 5.3 shows the result of using a sparse RBM [77].



(a) Feature detectors with high scores. (b) Feature detectors with low scores.

Figure 5.2: Feature detectors learned from RBM on MNIST dataset.

As can be seen, in Figure 5.2, high-scores are associated with visualisations of certain MNIST patterns, while low scores are mostly associated with fading or noisy



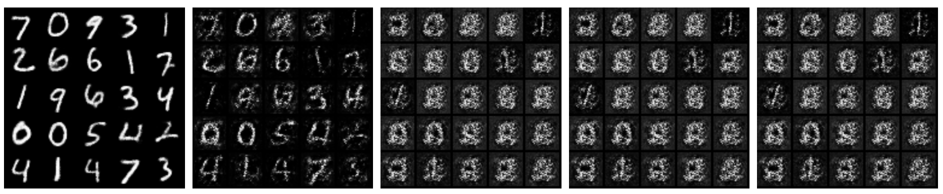
(a) Feature detectors with high scores. (b) Feature detectors with low scores.

Figure 5.3: Feature detectors from sparse RBM on MNIST dataset.

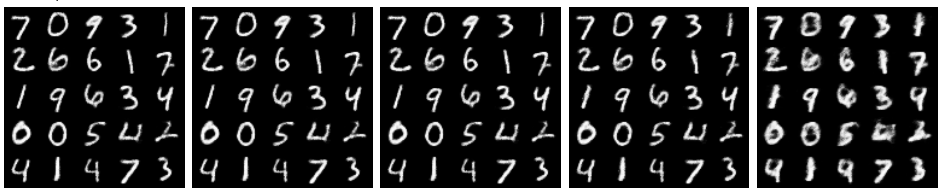
patterns. In Figure 5.3, high-scores are associated with sparse representations of the handwritten digits in the MNIST dataset, while low-scores produce less meaningful representations, e.g. digit 6 mixed up with digit 7.

5.2.4 Network Pruning

We also examined visually the impact of pruning using the confidence values on an RBM with 500 hidden units trained on the MNIST dataset with 10,000 examples. Sub-networks with the highest scores were gradually removed, and the pruned RBM was compared on the reconstruction of images with the original RBM, as illustrated in Figure 5.4a. In Figure 5.4b low-scoring feature detectors were gradually removed and the figure also shows the reconstruction of test images from the (pruned) RBMs.



(a) Reconstructed test images from RBM with high-scoring feature detectors pruned. From left to right, number of hidden units remaining: 500 (original RBM), 400 ,300, 200 and 100.



(b) Reconstructed test images from RBM with low-scoring feature detectors pruned. From left to right, number of hidden units remaining: 500 (original RBM), 400 ,300, 200 and 100.

Finally, in order to obtain accuracy measures, we have provided the features obtained from the pruned RBMs as input to an SVM classifier. Figure 5.5 shows the drop in accuracy with the gradual pruning of the RBM. In case of pruning low-scoring features, at first, the removal of units has produced a slight increase in accuracy. Then, the results indicate that it is possible to remove units and maintain performance almost unchanged until a point at which accuracy deteriorates, when more than half of the number of units is removed. In the case of pruning high-scoring features, the accuracy decreases significantly when 10% of the hidden units are removed. We repeated the experiments several times obtaining similar results.

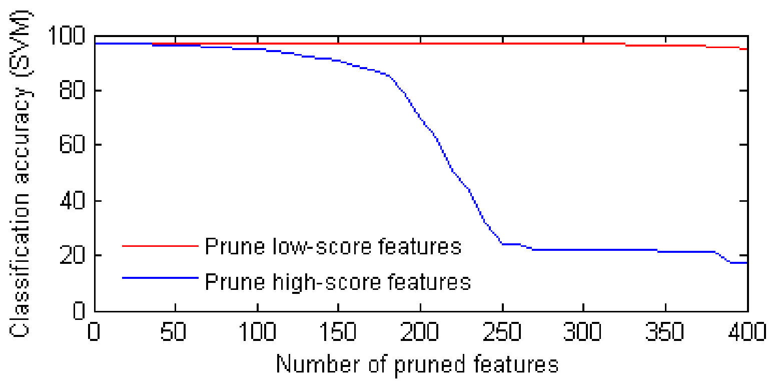


Figure 5.5: Classification accuracy of a pruned RBM, starting with 500 hidden units, on 10,000 MNIST test samples. The red and blue lines represent the accuracy following pruning of low-scoring and high-scoring feature detectors respectively.

One may see that the role of confidence values in ranking representations is similar to that of eigenvalues in PCA. For example, when pruning an RBM with $1,000$ hidden units trained on $10,000$ MNIST samples we obtain the best accuracy 97.54% on the test set with 600 best representations, while the whole RBM achieves 97.34% accuracy. We also performed similar experiments with PCA and observed that the highest accuracy is 97.16% with 100 best PCA features while the whole set of PCA features achieved 96.65% accuracy.

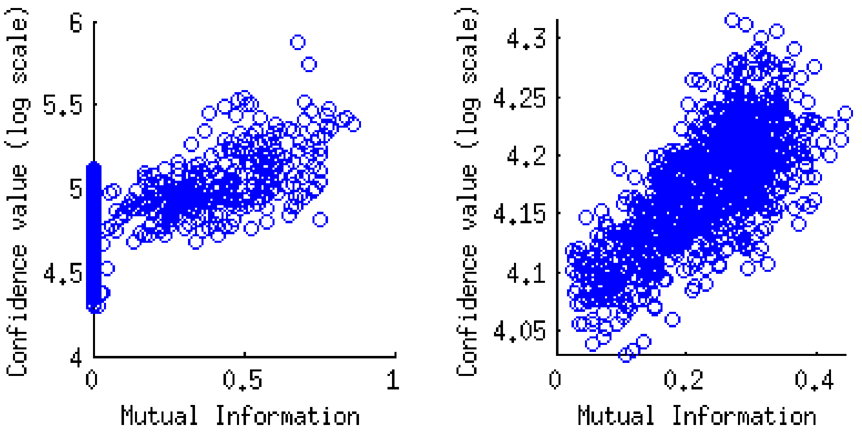
5.2.5 Mutual Information Measurement

In a statistical model, mutual information can be used to evaluate learning by comparing the dependency between the model distribution and the data distribution. In RBMs, by considering each feature detector (i.e. complete-model representing a sub-network) we can evaluate its usefulness by approximating the mutual information between a hidden variable (h_j to which the feature detector connects) and a visible variable [13], as follows.

$$\text{Score}_{MI}(\mathbf{w}_j) = \text{MI}(h_j, \mathbf{x}) = \sum_{h_j=0}^{1} \sum_{\mathbf{x}} p(h_j, \mathbf{x}) \log_2 \frac{p(h_j|\mathbf{x})}{p(h_j)} \tag{5.5}$$

Using mutual information is statistically sound, however it requires the use of training data to approximate $p(\mathbf{x})$. Our ranking approach, in contrast, is independent from the task that the model has been trained for.

We have measured how mutual information relates to our high and low-scoring complete models by measuring the statistical relevance of the ranked feature detectors using mutual information.



(a) 20000 training samples. (b) 60000 training samples.

Figure 5.6: Mutual Information measurement on ranked feature detectors.

We trained RBMs with 1000 hidden units on two different training sets, one consists of 20,000 samples and the other consists of 60,000 samples. We used SVMs as in the previous section to evaluate the accuracy obtained by each feature selection. We ranked the feature detectors in the best RBM (for each training set) and also measured their mutual information. Figure 5.6 shows the relationship between the confidence values and the mutual information measurements. The figures indicate that, in both cases, most of the detectors with higher confidence values are likely to have higher mutual information, although there is no formal proof for this relationship.

5.3 Representation Ranking for Knowledge Reuse

5.3.1 Self-taught Learning using Unlabelled Data

Unsupervised Transfer Learning with Sparse Coding

Self-taught learning [110] is a representation transfer learning method that has been applied to different target domains using an efficient learning algorithm for sparse coding [76]. Keeping the notation similar to that of the RBMs above, we use $W \in \mathbb{R}^{I \times J}$ to denote the J basis column vectors, and h to denote the sparse coder's coefficient vector (or feature vector). The basis vectors are learned by solving the following optimisation:

$$\begin{aligned} &\text{Minimise} \sum_{\mathbf{x}^{\mathcal{S}} \in \mathcal{D}^{\mathcal{S}}} ||\mathbf{x}^{\mathcal{S}} - \mathbf{W}\mathbf{h}^{\mathcal{S}}||_2^2 + \beta ||\mathbf{h}^{\mathcal{S}}||_1 \\ &s.t. \qquad ||\mathbf{w}_j||_2 \leq 1 \text{ for all } j \in J \end{aligned} \tag{5.6}$$

where $\mathbf{w}_j$ is a basis column vector j. The basis vectors $\mathbf{W}$ are then transferred to a target domain, as follows.

$$\mathbf{h}^{*\mathcal{T}} = \arg\max_{\mathbf{h}^{\mathcal{T}}} ||\mathbf{x}^{\mathcal{T}} - \mathbf{W}\mathbf{h}^{\mathcal{T}}||_2^2 + \beta ||\mathbf{h}^{\mathcal{T}}||_1 \tag{5.7}$$

The features are then used in a classification task with an improvement in performance expected due to generality [110].

Unsupervised Transfer Learning with RBMs

In general, self-taught learning with unlabelled data can be applied to any unsupervised learning model. In the case of RBMs, the basis vectors are called feature detector (the column vectors in the weight matrix), which can be learned by solving the following log-likelihood optimisation:

$$\text{Minimise} \quad -\frac{1}{N}\sum_{n} \log(\sum_{\mathbf{h}} P(\mathbf{x}^{\mathcal{S}}, \mathbf{h}^{\mathcal{S}})) \tag{5.8}$$

The approximation training can be performed using CD [52], and the complete-models transferred to the target domain, as follows:

$$\begin{aligned}\mathbf{h}^{*\mathcal{T}} &= \arg\max_{\mathbf{h}^{\mathcal{T}}} P(\mathbf{x}^{\mathcal{T}}, \mathbf{h}^{\mathcal{T}}) \\ &= \arg\max_{\mathbf{h}^{\mathcal{T}}} P(\mathbf{h}^{\mathcal{T}}|\mathbf{x}^{\mathcal{T}})P(\mathbf{x}^{\mathcal{T}})\end{aligned} \tag{5.9}$$

where $h_j^{*\mathcal{T}} = 1$ if $\sigma(\mathbf{w}_j^\top \mathbf{x}^{\mathcal{T}} + b_j) > 0.5$, otherwise, $h_j^{*\mathcal{T}} = 0$. However, in practice, it is common to use the more efficient $P(\mathbf{h}^{\mathcal{T}}|\mathbf{x}^{\mathcal{T}})$ directly as the features to train a classifier.

5.3.2 Learning with Guidance

In this section, we investigate how confidence values can be used to guide transfer learning. The general idea is to transfer the knowledge extracted from a domain to improve the learning in another domain. In this case, we can transfer the rules with the highest confidence values. Given an RBM trained on a dataset, we are interested in investigating whether the representation learned from a domain can be useful at improving the predictive representation learned by another RBM on a related, but different domain. We use $\theta^{\mathcal{S}}$ to denote knowledge extracted from a source RBM, and $\theta^{\mathcal{T}}$ to denote knowledge we want to learn from a target RBM. Since $\theta^{\mathcal{S}}$ is transferred onto the target RBM which is to learn $\theta^{\mathcal{T}}$, we expect the target RBM to learn additional knowledge θ^*. More formally, the target RBM is to learn $\theta^{\mathcal{T}}$ by combining learning from $\theta^{\mathcal{S}}$, which is fixed, and θ^*, as follows:

$$\text{Minimise} \quad \mathcal{L}^{\mathcal{T}} = -\frac{1}{M}\sum_m \log P(\mathbf{x}^{\mathcal{T}(m)}|\{\theta^{\mathcal{S}}, \theta^*\}) \tag{5.10}$$

In the case when one would like to treat $\theta^{\mathcal{S}}$ and θ^* as independent, as follows:

$$\begin{aligned}P(\mathbf{x}^{\mathcal{T}}|\theta^{\mathcal{S}}, \theta^*) &= \frac{P(\theta^{\mathcal{S}}|\mathbf{x}^{\mathcal{T}}, \theta^*)P(\mathbf{x}^{\mathcal{T}}, \theta^*)}{P(\theta^{\mathcal{S}}, \theta^*)} \\ &= \frac{P(\theta^{\mathcal{S}}|\mathbf{x}^{\mathcal{T}})P(\mathbf{x}^{\mathcal{T}}, \theta^*)}{P(\theta^{\mathcal{S}})P(\theta^*)} \\ &\propto P(\mathbf{x}^{\mathcal{T}}|\theta^{\mathcal{S}})P(\mathbf{x}^{\mathcal{T}}|\theta^*)\end{aligned} \tag{5.11}$$

The negative log-likelihood in Eq. 5.10 and the conditional probability in Eq. 5.11 tell us that if $\theta^{\mathcal{S}}$ and θ^{*} are independent then minimising the negative log-likelihood reduces to training an RBM in the target domain, i.e. minimising $-\log P(\mathbf{x}^{\mathcal{T}}|\theta^{*})$, since $\log P(\mathbf{x}^{\mathcal{T}}|\theta^{\mathcal{S}})$ is constant.

When $\theta^{\mathcal{S}}$ and θ^{*} are not independent, the learning of θ^{*} should be influenced by the presence of $\theta^{\mathcal{S}}$. We observe that when the whole set of complete-models learned in source RBMs is transferred, the transfer model seems to saturate, i.e. it is dominated by the transferred knowledge which causes very little additional knowledge to be learned. This happens because the dependency constrains the model to generate biased samples for learning with CD [54]. In order to reduce the number of complete-models to be transferred while preserving the knowledge that has been learned in the source domain, we only transfer a subset of the complete-models with high confidence values. We model this as shown in Figure 5.7, where we specify the source knowledge $\theta^{\mathcal{S}}$ as the weight sub-matrix $\mathbf{W}^{\mathcal{S}}$ selected from a source RBM, and the additional knowledge to be learned, θ^{*}, as a weight matrix $\mathbf{W}^{*}$; we omit the biases for ease of presentation.

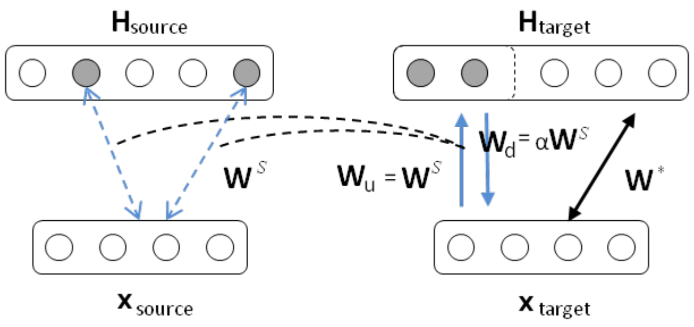

$\mathbf{H}_{source}$
$\mathbf{H}_{target}$
$\mathbf{W}^{\mathcal{S}}$
$\mathbf{W}_u = \mathbf{W}^{\mathcal{S}}$
$\mathbf{W}_d = \alpha\mathbf{W}^{\mathcal{S}}$
$\mathbf{W}^{*}$
$\mathbf{x}_{source}$
$\mathbf{x}_{target}$

Figure 5.7: General representation transfer model for unsupervised learning.

Knowledge from the source network is selected by ordering the scores (confidence values) obtained from Eq. 5.2, and transferring onto the target network, as explained in what follows. The connections between the visible layer and the hidden units transferred onto the target RBM can be seen as two sets of up-weights and down-weights. How much the down-weights affect the learning in the target RBM depends on the value of an influence factor $\alpha \in [0, 1]$. If $\alpha = 0$ then $\theta^{\mathcal{S}}$ and θ^{*} are independent, and the transferred knowledge will not influence learning in the target domain. Otherwise, if $\alpha = 1$ then the transferred knowledge will influence the learning in the target domain by using the source representation in the reconstruction of the target data during CD, as formally defined below. We call this *Guided Self-taught Learning* because the knowledge transferred from the source domain is used to guide the learning of representation in the target domain.

The energy function of the proposed learning model in Figure 5.7 (with $\alpha = 1$) is given by:

$$E(\mathbf{x}^{\mathcal{T}}, \mathbf{h}; \theta^{\mathcal{S}}, \theta^{*}) = E(\mathbf{x}^{\mathcal{T}}, \mathbf{h}_{*}; \theta^{*}) + E(\mathbf{x}^{\mathcal{T}}, \mathbf{h}^{\mathcal{S}}; \theta^{\mathcal{S}}) \tag{5.12}$$

As usual, we train the target RBM to minimise the negative log-likelihood Eq. 5.10 using CD. The gradient in the target RBM is:

$$\frac{\partial \mathcal{L}^{\mathcal{T}}}{\partial \theta^{*}} = \mathbb{E}\Big[\frac{\partial E(\mathbf{x}^{\mathcal{T}}, \mathbf{h}_{*}; \theta^{*})}{\partial \theta^{*}}\Big]_{\mathbf{h}_{*}|\mathbf{x}^{\mathcal{T}}} - \mathbb{E}\Big[\frac{\partial E(\mathbf{x}^{\mathcal{T}}, \mathbf{h}_{*}; \theta^{*})}{\partial \theta^{*}}\Big]_{\mathbf{h}_{*},\mathbf{h}^{\mathcal{S}},\mathbf{x}^{\mathcal{T}}} \tag{5.13}$$

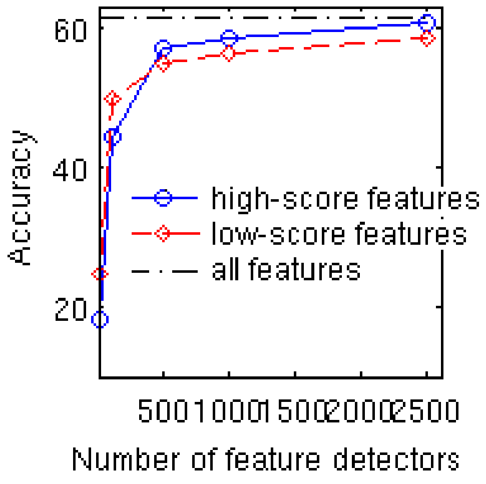
Although, at first sight, the above may seem similar to training the RBM on the target domain alone, the second term of the expectation must be approximated by sampling from the distribution $P(\mathbf{x}^{\mathcal{T}}, \mathbf{h}_{*}, \mathbf{h}^{\mathcal{S}})$. Through the sampling process, the transfer learning should capture the dependency of the target knowledge on the source, as is evaluated in the next Section. We detail the learning steps in Algorithm 7.

Algorithm 7 Guidance transfer learning algorithm

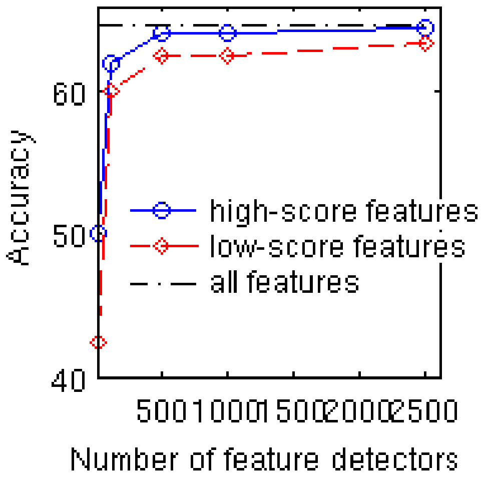
```
Require: A trained RBM: $N^{\mathcal{S}}$, MAX_ITER
 1: Select a number of sub-networks $\mathbf{W}^{\mathcal{S}} \in N^{\mathcal{S}}$ with the highest scores
 2: Encode $\mathbf{W}^{\mathcal{S}}$ into a new RBM: $N^{\mathcal{T}}$
 3: Add hidden units $H_{*}$ to $N^{\mathcal{T}}$ and create new parameters $\mathbf{W}^{*}$
 4: for $i = 1$ to MAX_ITER do
 5:     $\mathbf{X}_{pos} := input$ %start positive stage: assign data to visible layer.
 6:     $\mathbf{H}_{pos} := p(H|\mathbf{X}_{pos})$;     $\hat{\mathbf{H}}_{pos} \sim p(H|\mathbf{X}_{pos})$;
 7:     $\mathbf{X}_{neg} := p(X|\hat{\mathbf{H}}_{pos}; \alpha)$; % start negative phase: reconstruct the data with $\alpha$ is set to 0 or 1.
 8:     $\hat{\mathbf{X}}_{neg} \sim p(X|\hat{\mathbf{H}}_{pos}; \alpha)$     $\mathbf{H}_{neg} \leftarrow p(H|\hat{\mathbf{X}}_{neg})$
 9:     % Updating additional parameter
10:     $\mathbf{W}^{*} = \mathbf{W}^{*} + \eta(\langle \mathbf{X}_{pos}^{T}\mathbf{H}_{*pos} - \hat{\mathbf{X}}_{neg}^{T}\mathbf{H}_{*neg} \rangle)$
11: end for
12: % MAX_ITER is the number of training epoch which is sellected empirically, i.e. using the validation set.
13: return $\mathbf{W}^{\mathcal{T}} = \{\mathbf{W}^{\mathcal{S}}, \mathbf{W}^{*}\}$
14: %$\mathbf{W}^{\mathcal{T}}$ is used to extract features to learn a classifier as discussed in subsection 5.3.1.
```

5.3.3 Experimental Results

In order to evaluate transfer learning in the context of images, we have transferred features from an RBM trained on the TiCC collection; we denote as TiCC$_d$ and TiCC$_a$ the datasets of digits and letters, respectively. In order to show the effectiveness of reusing a trained network, we train RBMs in source domains and apply GSTL, Section 5.3, to target domains. The features learned from target domains then are used to train a SVM classifier. In all experiments, we use the SVM with Gaussian Kernel (i.e. $\mathcal{K} = \exp(-\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2)$). For each domain, the data is divided into training, validation and test sets. Each experiment is repeated 50 times and validation sets are used for model selection. The percentages show the predictive accuracy on the target domain, as detailed in the sequel.



(a) TiCC letters to TiCC digits.



(b) TiCC digits to TiCC letters.

Figure 5.8: Self-taught learning using RBMs where only a subset of features have been transferred.

	TiCC$_d$: TiCC$_a$	TICC$_a$: TiCC$_d$
SVM	59.16	60.34
RBM	62.85 ± 0.079	63.42 ± 0.090
SC STL	60.73	60.13
RBM STL	61.50 ± 0.125	64.71 ± 0.094
GSTL ($\alpha = 0$)	62.41 ± 0.166	**66.10 ± 0.137**
GSTL ($\alpha = 1$)	**63.16 ± 0.120**	**66.25 ± 0.175**

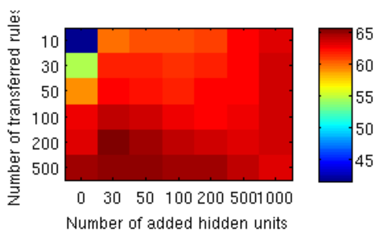
Table 5.3: Transfer learning experimental results for datasets in TiCC collection. The percentages show the average predictive accuracy on the target domain with 95% confidence interval.

For the RBMs in source domain we ranked the complete-models according to their confidence values and then gradually transferred the detectors with the highest scores. We also repeat the same process with the lowest ones. In this experiment the source RBMs consist of $5,000$ hidden units. The results of elf-taught learning with RBMs in Figure 5.8 show that by transferring only a half of the highest scoring detectors we can achieve similar performance as transferring all detectors.

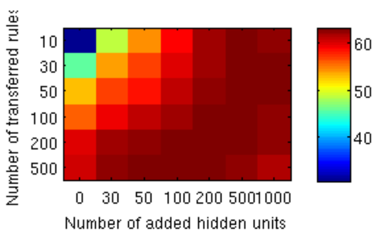
We compare GSTL with the baseline SVM, and the RBM features learned in the target domain (also using SVMs as classifier). We also compare GSTL with the self-taught learning using different models such as Sparse Coding[1] (SC STL), PCA (PCA STL), and RBM (RBM STL). The results in Table 5.3 show a consistent improvement of GSTL over both STL RBM and RBMs learned in target domains. In both cases, our approach shows better performance than the use of raw features (SVM), the RBM features, and the features from self-taught learning with Sparse Coding or RBMs. For a comparison between using the dependency constraint ($\alpha = 1$) and the mixture of features ($\alpha = 0$), the first experiment (where the representation learned from digits has been transferred to learn letters) shows a statistically significant improvement by applying the dependency constraint, while in the other case the results are slightly better but not significant.

With transfer, it is generally accepted that the performance of the model in a target domain will depend on the quality of the knowledge it received and the structure of the model. We evaluated performance of the model using different numbers of transferred complete-models and number of units added to the hidden layer. Figure 5.9 shows that if the number of transferred complete-models is too small, it will be dominated by the data from the target domain. However, if the number of transferred complete-models is much larger than the additional knowledge it can cause a drop in performance since the model will try to learn new knowledge mainly based on the transferred knowledge with little knowledge from the target domain.

[1]http://ai.stanford.edu/~hllee/softwares/nips06-sparsecoding.htm



(a) TiCC letters to TiCC digits.



(b) TiCC digits to TiCC letters.

Figure 5.9: Performance of learning with guidance for different numbers of transferred complete-models and additional hidden units. The colour-bars map accuracy to the colour of the cells as shown so that the hotter the colour, the higher the accuracy.

5.4 Summary

This chapter presented an efficient method to compute confidence values for the purpose of ranking representations in RBMs. The hypothesis behind this is that the confidence value of a rule is capable of representing the confidence of the representation it was extracted from. Similar as rule extraction, we showed that high ranking representation are able to capture the majority of the network, and therefore be useful for network pruning to achieve more compactness while preserving the performance. We also showed that the complete-models with high confidence values are more likely to be relevant with the data than the ones with low confidence values, based on mutual information measurement.

From the idea of representation ranking we further showed the usefulness of high-ranking representations in self-taught learning. In this case the transferred knowledge can be seen as a set of complete-models with highest confidence values. The experiment results showed that using only representations with highest confidence values would give the same performance as transferring the whole network. Furthermore, a small number of highly confident representations can also be used to guide the learning of useful features in the target RBMs.

Chapter 6

Adaptive Transferred Profile Likelihood Learning

The previous chapter introduced a method to use confidence values to rank and transfer complete-models. A complete-model can be seen as a feature detector (a column vector of the weight matrix) associated with a confidence value. The complete-models are obtained from an RBM to be transferred and improve the representation learning in another (target) domain. However, this method does not take into account the fact that the feature detectors learned in the source domain may not be useful for learning in the target domain. In order to address this issue, this chapter proposes a framework for transforming and adapting the source knowledge onto the target domain.

6.1 Motivation

Chapter 5 discussed a variant of self-taught learning [110, 80] in that the representation learned from a source domain is reused by being transferred to a target domain to improve the representation learning. Even though the representation learned from a large amount of source data can help extract more general features from the target data, there exists a problem that the representation learned from the source domain may contain knowledge that is not useful with the target domain data [101]. In order to address this potential problem we can treat the source representation as a set of independent *complete-models*, and associate each of them with adaptive factors for integration into an unsupervised learning model in the target domain. Differently from other methods [110, 80] which only use the source representation, our model has what we call supplementary knowledge added alongside the transferred knowledge. We learn the adaptive factors and the supplementary knowledge parameters by maximising log-likelihood while keeping the transferred knowledge fixed. We name this method *adaptive transferred profile likelihood* because it is inspired by the idea of profile likelihood [35]. During learning, in order to maximise the model's likelihood over the data, the adaptive factors will transform the transferred feature detectors such that they can hopefully improve their effectiveness in the new domain.

In this transfer learning approach we extend the model proposed in §5.3.2 to let the transferred representations be transformed (in a precise sense to be defined below) while at the same time guiding the learning of new representations in the target domain. To this end we associate each representation $\mathbf{w}$ to be transferred with a factor λ to create a transformed presentation $\mathbf{w}' = \lambda \times \mathbf{w}$ in the target domain. The transferred representations are fixed while the factors are adjusted so that in the end, after being transformed, the new representations are expected to be more closely related to the target data. Furthermore, similar to what has been done in §5.3.2 we also add new biases to the hidden units created by transferred representations. This makes a transferred hidden feature $h = \sigma(\mathbf{x}^T(\lambda \times \mathbf{w}) + b)$ a sigmoid nonlinear function of a fixed transferred representation with adjustable factors λ and b learned from the target domain.

Example 6.1.1. Figure 6.1 illustrates an example of transferring the representation from Plus image to Minus image. The Plus image is learned by an RBM from which we can decompose the weight matrix into a set of feature detectors representing some parts of the image. Some selected feature detectors (associating with h_1 and h_2) are transferred to the target domain where the first detector (associating with h_1) will be assigned a zero adaptation factor ($\lambda_1 = 0.0$) while the second one (associating with h_2) is assigned a positive adaptation factor ($\lambda_2 = 1.0$). These assignments are defined based on the fact that the first feature detector does not bear any related knowledge about the Minus sign. In contrast, the second detector presents a part of the Minus sign so that it should be useful in the target domain.

For completeness, new knowledge from the target domain should also be learned as the supplementary to the transferred one. We expect that aTPL framework can produce the similar effect by learning adaptation factors automatically, the details will be presented in §6.2.

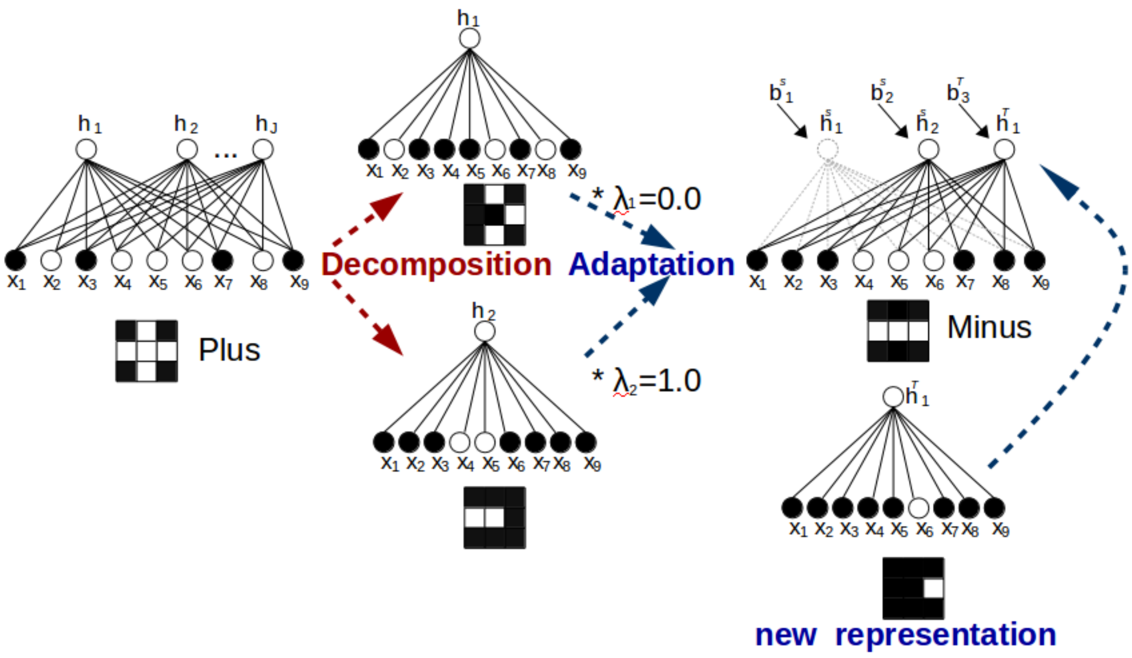


Figure 6.1: aTPL example of transferring representations from Plus image to Minus image.

6.2 Representation Transfer and Adaptation

In this section, first the Profile Likelihood idea is recalled. Then, it is applied to the problem of transfer learning: restricted Boltzmann machines are adopted for unsupervised learning in both the source and target domains. Adaptation is added by training the RBM on the target domain and using the adaptive factors on the transferred representations.

6.2.1 Profile Likelihood

Maximum Likelihood Estimation (MLE) is a popular method for fitting a statistical model to a data distribution. Given a training data set $\mathcal{D}$, the MLE will train a model parameterised by Θ such that it maximises the likelihood:

$$\mathcal{L}(\Theta; \mathcal{D}) = \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}|\Theta) \tag{6.1}$$

Suppose that we are interested in a subset of the parameters, say θ^c, from the whole set of parameters $\Theta = \{\theta^c, \theta^f\}$. One can estimate such parameters of interest by profiling out the nuisance parameters θ^f and maximising the following profile likelihood [35]:

$$\mathcal{L}_{\hat{\theta}^f}(\theta^c; \mathcal{D}) = \mathcal{L}(\theta^c; \hat{\theta}^f, \mathcal{D}) \tag{6.2}$$

where $\hat{\theta}^f = \arg\max_{\theta^f} \mathcal{L}(\theta^f; \mathcal{D}, \theta^c)$ is a function of θ^c.

6.2.2 Proposed Model

Consider a special case of the profile likelihood where the nuisance parameters are constant, i.e. θ^f are independent from θ^c. In particular, θ^f can be seen as estimates from a related, but different domain $\mathcal{S}$, as follows: $\hat{\theta}^f = \arg\max_{\theta^f} \mathcal{L}(\theta^f; \mathcal{D}^{\mathcal{S}})$. Let us estimate the other parameters with adaptation factors Λ and Ψ, as follows:

$$\mathcal{L}_{\hat{\theta}^f}(\theta^c, \Lambda, \Psi; \mathcal{D}) = \mathcal{L}(\theta^c, \Lambda, \Psi; \hat{\theta}^f, \mathcal{D}) = \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}|\theta^c, \Lambda \circ \hat{\theta}^f, \Psi) \tag{6.3}$$

Here, the value of Λ and Ψ will decide how $\hat{\theta}^f$ should adapt to the new domain: Λ applies directly to the representation transferred through an element-wise product, denoted by $\circ$, while Ψ influences the learning indirectly, as will be explained next.

In representation learning, the knowledge learned from a domain is normally denoted by the model's parameters. We denote the representation knowledge which has been learned from a source domain $\mathcal{S}$ as $\mathbf{W}^{\mathcal{S}} = \{\mathbf{w}_{j'} \in \mathbb{R}^I | j' = 1, .., J^{\mathcal{S}}\}$; $\mathbf{W}^{\mathcal{S}}$ is known as a set of feature detectors, also called basis vectors [100, 76]). Each feature detector $\mathbf{w}_j^{\mathcal{S}}$ is a column vector of the weight matrix $\mathbf{W}^{\mathcal{S}}$. Our objective is to transform each feature detector $\mathbf{w}_j^{\mathcal{S}}$ to adapt to a target domain $\mathcal{T}$ while learning takes place in the target domain producing new feature detectors $\mathbf{W}^{\mathcal{T}} = \{\mathbf{w}_{j''} \in \mathbb{R}^I | j'' = 1, .., J^{\mathcal{T}}\}$.

An RBM as an unsupervised statistical model with observed variable $\mathbf{x}$ and hidden variable $\mathbf{h}$ presents an energy-based distribution:

$$P(\mathbf{x}|\Theta) = \frac{1}{Z}\sum_{\mathbf{h}} \exp^{-E(\mathbf{x},\mathbf{h},\Theta)}$$

where $Z = \sum_{\mathbf{x},\mathbf{h}} \exp^{-E(\mathbf{x},\mathbf{h},\Theta)}$ and $\Theta = \{\mathbf{W} \in \mathbb{R}^{I\times J}, \mathbf{a} \in \mathbb{R}^{I}, \mathbf{b} \in \mathbb{R}^{J}\}$ with $\mathbf{W} = \{\mathbf{w}_j \in \mathbb{R}^{I} | j = 1, ..J\}$, and an associated energy function:

$$E(\mathbf{x},\mathbf{h},\Theta) = -\sum_{j=1}^{J} h_j(\mathbf{x}^\top \mathbf{w}_j) - \mathbf{x}^\top \mathbf{a} - \mathbf{h}^\top \mathbf{b} \tag{6.4}$$

Here, $\mathbf{a}$ and $\mathbf{b}$ are the biases associated with observed and hidden variables, respectively. Suppose that a learned representation $\{\mathbf{W}^{\mathcal{S}}\}$ is provided. We are interested in reusing this knowledge in domain $\mathcal{T}$ by forming the joint parameters $\Theta = \{\mathbf{W} = \{\mathbf{W}^{\mathcal{S}}, \mathbf{W}^{\mathcal{T}}\}, \mathbf{a}, \mathbf{b}^{\mathcal{T}}\}$ to model a distribution $P(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \mathbf{h}^{\mathcal{T}}|\Theta)$ with:

$$\begin{aligned} E(\mathbf{x},\mathbf{h}^{\mathcal{S}},\mathbf{h}^{\mathcal{T}},\Theta) = &-\sum_{j'=1}^{J^{\mathcal{S}}} h^{\mathcal{S}}_{j'}(\mathbf{x}^\top \mathbf{w}^{\mathcal{S}}_{j'}) - \sum_{j''=1}^{J^{\mathcal{T}}} h^{\mathcal{T}}_{j''}(\mathbf{x}^\top \mathbf{w}^{\mathcal{T}}_{j''}) \\ &- \mathbf{x}^\top \mathbf{a} - \mathbf{h}^{\mathcal{T}\top} \mathbf{b}^{\mathcal{T}} \end{aligned} \tag{6.5}$$

Notice that since only the feature detectors have been transferred, $\mathbf{b}^{\mathcal{T}} \in \mathbb{R}^{J^{\mathcal{T}}}$ denote the biases of hidden variable $\mathbf{h}^{\mathcal{T}}$. As discussed earlier, in this model, $\hat{\theta}^f = \{\mathbf{W}^{\mathcal{S}}\}$ are fixed parameters, while $\theta^c = \{\mathbf{W}^{\mathcal{T}}, \mathbf{a}, \mathbf{b}^{\mathcal{T}}\}$ are adjustable. Furthermore, adaptation factors are associated with each feature detector $\mathbf{w}_{j'}$ according to Eq. Eq. 6.3, so that the energy function becomes:

$$\begin{aligned} E(\mathbf{x},\mathbf{h}^{\mathcal{S}},\mathbf{h}^{\mathcal{T}},\Theta) &= E(\mathbf{x},\mathbf{h}^{\mathcal{S}},\mathbf{h}^{\mathcal{T}},\theta^c, \Lambda \circ \hat{\theta}^f, \Psi) \\ &= E(\mathbf{x},\mathbf{h}^{\mathcal{S}},\hat{\theta}^f, \lambda, \psi) + E(\mathbf{x},\mathbf{h}^{\mathcal{T}},\theta^c) \end{aligned} \tag{6.6}$$

where $\Lambda = \begin{bmatrix} \lambda^\top \\ ... \\ \lambda^\top \end{bmatrix} \in \mathbb{R}^{I\times J^{\mathcal{S}}}$ are direct adaptation factors and $\psi = \Psi \in \mathbb{R}^{J^{\mathcal{S}}}$ are the indirect adaptation factors. The first term in Eq. 6.6 denotes the energy of the adaptive transfer, as follows:

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \hat{\theta}^f, \boldsymbol{\lambda}, \boldsymbol{\psi}) &= -\sum_{j'=1}^{J^{\mathcal{S}}} h_{j'}^{\mathcal{S}}(\mathbf{x}^\top(\lambda_{j'} \times \mathbf{w}_{j'}^{\mathcal{S}}) + \psi_{j'}) \\ &= -\sum_{j'=1}^{J^{\mathcal{S}}} \lambda_{j'} h_{j'}^{\mathcal{S}}(\mathbf{x}^\top \mathbf{w}_{j'}^{\mathcal{S}}) + \mathbf{h}^{\mathcal{S}^\top} \boldsymbol{\psi} \end{aligned} \tag{6.7}$$

From Eq. 6.7, one can see that the adaptive factors $\boldsymbol{\psi}$ are added to the biases of the hidden variable $\mathbf{h}^{\mathcal{S}}$. Differently from λ_j, the adaptive factor ψ_j affects the transferred feature detectors indirectly depending on the input $\mathbf{x}$ and the feature detector $\mathbf{w}_j^{\mathcal{S}}$. The probability of activation of hidden variable h_j is given by a sigmoid function $P(h_j|\mathbf{x}, \Theta) = \sigma(\lambda_j(\mathbf{x}^\top \mathbf{w}_j^{\mathcal{S}}) + \psi_j)$. In the case that the value of ψ_j is large then the feature detector is removed from the presentation. Together with the second energy term in Eq. 6.6: $E(\mathbf{x}, \mathbf{h}^{\mathcal{T}}, \theta^c) = -\sum_{j''=1}^{J^{\mathcal{T}}} h_{j''}^{\mathcal{T}}(\mathbf{x}^\top \mathbf{w}_{j''}^{\mathcal{T}}) - \mathbf{x}^\top \mathbf{a} - \mathbf{h}^{\mathcal{T}^\top} \mathbf{b}^{\mathcal{T}}$, the energy function of the target RBMs finally becomes:

$$E(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \mathbf{h}^{\mathcal{T}}, \Theta) = -\sum_{j'=1}^{J^{\mathcal{S}}} \lambda_{j'} h_{j'}^{\mathcal{S}}(\mathbf{x}^\top \mathbf{w}_{j'}^{\mathcal{S}}) + \mathbf{h}^{\mathcal{S}^\top} \boldsymbol{\psi} - \sum_{j''=1}^{J^{\mathcal{T}}} h_{j''}^{\mathcal{T}}(\mathbf{x}^\top \mathbf{w}_{j''}^{\mathcal{T}}) - \mathbf{x}^\top \mathbf{a} - \mathbf{h}^{\mathcal{T}^\top} \mathbf{b}^{\mathcal{T}} \tag{6.8}$$

6.2.3 Learning

We train the model by maximising the following adaptive profile log-likelihood:

$$\begin{aligned} \log \mathcal{L}(\theta^c, \boldsymbol{\lambda}, \boldsymbol{\psi}; \hat{\theta}^f, \mathcal{D}) = &\sum_x \log \sum_{\mathbf{h}^{\mathcal{S}}, \mathbf{h}^{\mathcal{T}}} \exp(-E(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \mathbf{h}^{\mathcal{T}}, \Theta)) \\ &- \log Z \end{aligned} \tag{6.9}$$

where the energy function of the target RBM is in Eq. 6.8.

The gradients of the adaptive profile log-likelihood Eq. 6.9 with respect to the supplementary parameters and the adaptation factors are estimated as:

$$\begin{aligned} \frac{\partial \log \mathcal{L}(\theta^c, \boldsymbol{\lambda}, \boldsymbol{\psi}; \hat{\theta}^f, \mathcal{D})}{\partial \theta^c} = &\mathbb{E}\Big[\frac{\partial E(\mathbf{x}, \mathbf{h}^{\mathcal{T}}, \theta^c)}{\partial \theta^c}\Big]_{\mathbf{h}^{\mathcal{T}}|\mathbf{x}} \\ &- \mathbb{E}\Big[\frac{\partial E(\mathbf{x}, \mathbf{h}^{\mathcal{T}}, \theta^c)}{\partial \theta^c}\Big]_{\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \mathbf{h}^{\mathcal{T}}} \end{aligned} \tag{6.10}$$

$$\begin{aligned}\frac{\partial \log \mathcal{L}(\theta^c, \boldsymbol{\lambda}, \boldsymbol{\psi}; \hat{\theta}^f, \mathcal{D})}{\partial \theta_{\boldsymbol{\lambda},\boldsymbol{\psi}}} &= \mathbb{E}\Big[\frac{\partial E(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \hat{\theta}^f, \boldsymbol{\lambda}, \boldsymbol{\psi})}{\partial \theta_{\boldsymbol{\lambda},\boldsymbol{\psi}}}\Big]_{\mathbf{h}^{\mathcal{S}}|\mathbf{x}} \\ &- \mathbb{E}\Big[\frac{\partial E(\mathbf{x}, \mathbf{h}^{\mathcal{S}}, \hat{\theta}^f, \boldsymbol{\lambda}, \boldsymbol{\psi})}{\partial \theta_{\boldsymbol{\lambda},\boldsymbol{\psi}}}\Big]_{\mathbf{x},\mathbf{h}^{\mathcal{S}},\mathbf{h}^{\mathcal{T}}}\end{aligned} \tag{6.11}$$

Here, the gradient computations become intractable, and a commonly used Markov Chain Monte Carlo to collect a large number of samples of $\mathbf{x}$, $\mathbf{h}^{\mathcal{S}}$ and $\mathbf{h}^{\mathcal{T}}$, which, however, will be very computationally expensive. Therefore, the Contrastive Divergence approach [52] was followed to estimate the gradient computations efficiently, as follows:

$$\begin{aligned}&\nabla w^{\mathcal{T}}_{ij''} = \langle x_i h^{\mathcal{T}}_{j''}\rangle_0 - \langle x_i h^{\mathcal{T}}_{j''}\rangle_K \\ &\nabla a_i = \langle x_i\rangle_0 - \langle x_i\rangle_K \\ &\nabla b^{\mathcal{T}}_{j''} = \langle h^{\mathcal{T}}_{j''}\rangle_0 - \langle h^{\mathcal{T}}_{j''}\rangle_K \\ &\nabla \lambda_{j'} = \beta\Big(\langle h^{\mathcal{S}}_{j'} \sum_i x_i w^{\mathcal{S}}_{ij'}\rangle_0 - \langle h^{\mathcal{S}}_{j'} \sum_i x_i w^{\mathcal{S}}_{ij'}\rangle_K\Big) \\ &\nabla \psi_{j'} = \langle h^{\mathcal{S}}_{j'}\rangle_0 - \langle h^{\mathcal{S}}_{j'}\rangle_K\end{aligned} \tag{6.12}$$

Where K is a number of Gibbs sampling steps. Value of β will decide the adaptation rate the source knowledge in the target domain. In the case that $\boldsymbol{\lambda}$ is initialised to $1'$s and $\beta = 0$ then the adaptation is done according to $\boldsymbol{\psi}$ only. In the experiment section, we discuss the effectiveness of β. The detail of the learning algorithm is shown in Algorithm 8.

6.3 Biased Sampling

Transferring the entire set of feature detectors from source domain can be counter-productive and lead to lower accuracy. This is because samples from representations transferred can make learning of new knowledge too dependent on the source domain. This is similar as the co-adaptation effect described by Hinton [58]. In order to deal with this, a subset of feature detectors is selected from the source model to transfer. In **Chapter 5**, we use the confidence values to rank the feature detectors and transfer the highest scoring detectors. Another method is to rank and select a feature detector by *mutual information* (MI) between the hidden variable to which the feature detector connects to and the target samples. One also could apply the dropout technique [58, 127] by which the entire set of feature detectors

Algorithm 8 Adaptive Transferred-Profile Likelihood

```
Require: A set of learnt representation knowledge : $\mathbf{W}^{\mathcal{S}}$, MAX_ITER
 1: Initialise $\mathbf{W}^{\mathcal{T}}$, $\mathbf{a}$,$\mathbf{b}$, set $\boldsymbol{\lambda} = \vec{1}$ and $\boldsymbol{\psi} = \vec{0}$
 2: for $i = 1$ to MAX_ITER do
 3:     % start positive stage:
 4:     $\mathbf{X}_p := \mathbf{X}$ % assign batch data X
 5:     $\mathbf{H}^{\mathcal{S}}_p := p(\mathbf{H}^{\mathcal{S}}|\mathbf{X}); \hat{\mathbf{H}}^{\mathcal{S}}_p \sim p(\mathbf{H}^{\mathcal{S}}|\mathbf{X}_p);$
 6:     $\mathbf{H}^{\mathcal{T}}_p := p(\mathbf{H}^{\mathcal{T}}|\mathbf{X}); \hat{H}^{\mathcal{T}}_p \sim p(\mathbf{H}^{\mathcal{T}}|\mathbf{X}_p);$
 7:     % start negative phase
 8:     $\mathbf{X}_n := p(\mathbf{X}|\hat{\mathbf{H}}^{\mathcal{S}}_p, \hat{\mathbf{H}}^{\mathcal{T}}_p); \hat{\mathbf{X}}_n \sim p(\mathbf{X}|\hat{\mathbf{H}}^{\mathcal{S}}_p, \hat{\mathbf{H}}^{\mathcal{T}}_p);$
 9:     $\mathbf{H}^{\mathcal{S}}_n := p(\mathbf{H}^{\mathcal{S}}|\mathbf{X}); \mathbf{H}^{\mathcal{T}}_n := p(\mathbf{H}^{\mathcal{T}}|\mathbf{X});$
10:     % Updating supplementary parameters
11:     $\mathbf{W}^{\mathcal{T}} = \mathbf{W}^{\mathcal{T}} + \eta \times (\langle {\mathbf{X}_p}^{\top} {\mathbf{H}^{\mathcal{T}}}_p - \hat{\mathbf{X}}_n^{\top} \mathbf{H}^{\mathcal{T}}_n \rangle);$
12:     $\mathbf{a} = \mathbf{a} + \eta \times (\langle \mathbf{X}_p - \hat{\mathbf{X}}_n \rangle);$
13:     $\mathbf{b}^{\mathcal{T}} = \mathbf{b}^{\mathcal{T}} + \eta \times (\langle \mathbf{H}^{\mathcal{T}}_p - \mathbf{H}^{\mathcal{T}}_n \rangle);$
14:     % Updating adaptive factors
15:     $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \beta \times \eta \times (\langle \mathbf{H}^{\mathcal{S}}_p \circ ({\mathbf{W}^{\mathcal{S}}}^{\top} \mathbf{X}_p) - \mathbf{H}^{\mathcal{S}}_n \circ ({\mathbf{W}^{\mathcal{S}}}^{\top} \mathbf{X}_n))$
16:     $\boldsymbol{\psi} = \boldsymbol{\psi} + \eta \times (\langle \mathbf{H}^{\mathcal{S}}_p - \mathbf{H}^{\mathcal{S}}_n \rangle);$
17: end for
18: % MAX_ITER is the number of training epoch which is sellected empirically, i.e. using the validation set.
19: Set $\mathbf{w}^{\mathcal{S}}_{j'} = \lambda_{j'} \times \mathbf{w}^{\mathcal{S}}_{j'}$ for all $j'$ return $\Theta = \{\mathbf{W} = \{\mathbf{W}^{\mathcal{S}}, \mathbf{W}^{\mathcal{T}}\}, \mathbf{a}, \mathbf{b} = \{\boldsymbol{\psi}, \mathbf{b}^{\mathcal{T}}\}\}$
```

is transferred but only a subset of it is selected randomly for use at each learning batch. In the experiment section we will analyse the effect of all three approaches.

6.4 Experiments

In this section we empirically investigate the effectiveness of aTPL approach on image datasets. We compare the accuracy of SVMs trained on features from aTPL with the raw features (image pixels), the features from self-taught learning and the combined features from the source and the target domains. We also test how aTPL performs in the case of negative transfer.

6.4.1 Experimental Setting

Datasets

We performed the experiments on MNIST, TiCC, ICDAR, USPS, MADBASE data sets. These data sets have been widely used as benchmarks in feature learning and transfer learning [110, 1].

MNIST$_{30K}$: ICDAR$_d$ & TiCC$_w$. The MNIST dataset[1] consists of $60,000$ images of handwritten digits. The size of each images is 28×28 pixels. In these experiments the source representation is learnt from $30,000$ samples (MNIST$_{30k}$). In the target domains, we use the natural images of the ICDAR data set[2] for the digit recognition task with 10 classes, and the handwritten images of TiCC [3] for the writer recognition task with 5 classes. We partition the ICDAR data into 40% for training, 30% for validation, and 30% for testing. The TiCC writer data (TiCC$_w$) is partitioned into 500, 200, 200 samples for training, validation and testing, respectively.

MNIST$_{5K}$:TiCC$_d$ & TiCC$_a$. The MNIST$_{5k}$ contains 5000 samples from the MNIST dataset. The first task is to recognise the handwritten digits TiCC$_d$(10 classes) consisting of 30 training, 1000 validation, and 1500 test samples, extracted from the TiCC collection. The second task is to recognise the handwritten letters TiCC$_a$ with 10 samples for each class in the training set, the validation and test sets both consists of 3750 samples, also from the TiCC collection.

USPS[4]: MADBase[5]. In this experiment, the source domain is the handwritten digit dataset USPS, and the target domain is the handwritten Hindi digits. We divide the target data into 60 samples for training, 1000 for validation and 2000 for testing.

[1]http://yann.lecun.com/exdb/mnist/
[2]http://algoval.essex.ac.uk:8080/icdar2005/index.jsp?page=ocr.html
[3]http://homepage.tudelft.nl/19j49/Datasets.html
[4]http://www-i6.informatik.rwth-aachen.de/ keysers/usps.html
[5]http://datacenter.aucegypt.edu/shazeem/

Baselines

For completeness, we compare our methods with other methods that use representations learned from the source domain for transfer. In all experiments, we use SVM with a Gaussian Kernel (i.e. $\mathcal{K} = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2)$ as the classifier.

- SVM: There is no transfer from the source domain, the SVM classifier is learned directly from the target data.
- RBM: There is no transfer from the source domain, we train an RBM[6] on the target data and use the latent features to learn the SVM classifier.
- SC STL: The representations are transferred from the source domain using the self-taught learning approach with Sparse coding. The extracted features then is used for training the SVM classifier.
- RBM STL: The representations are transferred from the source domain using the self-taught learning approach with RBMs.
- RBM MIX: The representations learned by an RBM is transferred from the source domain and combined with the representations in the target domain also learned by another RBM.

For model selection we applied the grid-like search on the validation sets[7]. In our adaptive transferred profile likelihood (aTPL) model, we reuse the representation knowledge extracted from the self-taught RBM. In order to test the compactness of the model and also to reduce the computational cost in the testing phase, we only transfer a part of the complete-models. Therefore, we select T out of $J^{\mathcal{S}}$ complete-models by ranking them based on the confidence values in descending order, according to [138, 127], for the reuse. In this work, we select $T \leq \min\{500, J^{\mathcal{S}}\}$. The number of supplementary complete-models $J^{\mathcal{T}}$ are added such that the capacity of aTPL must be at most as large as the RBM STL (and smaller than RBM MIX).

[6]The RBMs are trained with sparse constraints following [77]

[7]For SVM, we search all hyper-parameters (cost and γ) from 0.0001 to 1000 and in a log scale. For the sparse coding, the number of bases are selected in {20, 50, 100, 150, 200, 500, 1000}, and the sparsity cost is searched from 0.001 to 100 in a log-scale. We select the RBMs by searching through the number of hidden unit in {50, 100, 500, 1000, 2000} and the learning rate in {0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7}, the sparsity cost is searched from 0.001 to 1000 and the sparsity level from 0.00001 to 0.1 in log scale. In all cases, if the optima is not apparent then the search is expanded.

In all experiments, we repeat 30 times and report the average results with a 95% confidence interval.

6.4.2 Experimental Results

	MNIST_{30k} : ICDAR_d	MNIST_{30k} : TiCC_w	MNIST_{05k} : TiCC_a	MNIST_{05k} : TiCC_d	USPS : MADBASE
SVM	39.04	73.44	59.16	60.34	80.4
RBM	37.63± 0.505	75.20 ± 0.745	62.85 ± 0.079	63.42 ± 0.090	80.38 ± 0.120
SC STL	46.23	70.06	55.82	57.78	81.7
RBM STL	52.26 ± 0.331	72.88 ± 0.098	58.13 ± 0.205	62.08 ± 0.321	81.43 ± 0.211
RBM MIX	52.43± 0.132	76.49 ± 0.361	63.21 ± 0.134	65.04 ± 0.330	80.90 ± 0.253
aTPL ($\beta = 0$)	51.64 ± 0.384	77.56 ± 0.564	63.00 ± 0.160	**65.75 ± 0.110**	**83.607 ± 0.151**
aTPL ($\beta = 0.01$)	**52.32 ± 0.347**	**80.45 ± 0.319**	**63.86 ± 0.185**	65.66 ± 0.122	83.11 ± 0.173

Table 6.1: Transfer learning experimental results: each column indicates a transfer experiment, e.g. MNIST_{30k}:ICDAR_d uses the MNIST handwritten digits (with 30,000 samples) as the source domain and the natural digit images ICDAR as the target domain. The percentages show the average predictive accuracy on the target domain with a 95% confidence interval. Results for SVMs are provided as a baseline. For the "SVM" and "RBM" lines, there is no transfer. The bold number indicates a statistically significant improvement. If the improvement is not apparent, then it indicate more compactness in terms of the model's capacity.

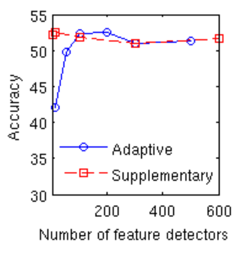
We present the experimental results in four transfer scenarios in Table 6.1. Each column indicates a representation reuse in the target domain where the representation has been learned in the source domain. We use a bold number to indicate the statistical significance. If the accuracy improvement is not apparent, we give the preference to the more compact one, i.e. the model with the smaller number of feature detectors.

The results in Table 6.1 show that self-taught transfer does not always work, especially when the training data in target domain is adequately large (MNIST_{30k}:TiCC_w) or the source domain is small (MNIST_{05k}:TiCC_a). One can see that, combining self-taught learning and representation learning in the target domain can somehow solve the problem. It makes sense because the additional representation learned from the target domain can reduce the negative effect of unsuitable transferred representation. With the adaptive transfer approach, we can not only further rule out the incompatible transferred representation but also enhance the useful ones. As a result, aTPL can significantly improve the performance of the classifiers in all five recognition tasks. An interesting remark which we will investigate further in the next section is that the improvement can be achieved by adding a small number of supplementary sub-networks.

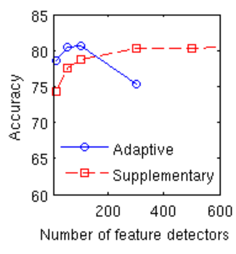
For completeness we also investigate the effectiveness of aPTL with the use of Mutual Information to select the complete-models for transfer and the use of the dropout technique as discussed in Section §6.3. The performance of MI in five scenarios MNIST$_{30k}$:ICDAR, MNIST$_{30k}$:TiCC$_w$, MNIST$_{05k}$:TiCC$_a$, MNIST$_{05k}$:TiCC$_d$ and USPS:MADBASE is 51.75 ± 0.322, 80.178 ± 0.249, 63.68 ± 0.145, 65.55 ± 0.274, 83.33 ± 0.303 respectively. These results are slightly lower than the use of confidence values due to the lack of training samples in the target domain which leads to poor approximation of the distribution over hidden variables. Moreover, the MI method is less efficient than the confidence values. The dropout technique achieves similar results to the confidence values in two cases MNIST$_{30k}$:TiCC$_w$ and MNIST$_{05k}$:TiCC$_d$, and slightly lower result in MNIST$_{05k}$:TiCC$_a$. Interestingly, in MNIST$_{30k}$:ICDAR and USPS:MADBASE it achieves **54.20 ± 0.233** and **84.08 ± 0.115** which are significantly better than the confidence values. However, dropout is the least efficient compared to confidence values and MI.

6.4.3 Adaptive vs. Supplementary Knowledge

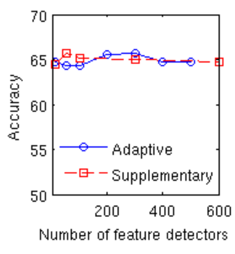
In this section, we investigate the relation of the adaptive knowledge and the supplementary knowledge in our model. We do this by gradually increasing the size (T) of the reused representation and in each case we gradually increase the size of the supplementary representation. Figure 6.2 shows that in most of the transfer



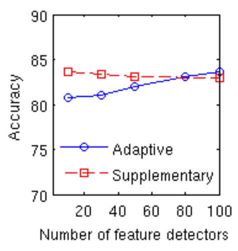
(a) MNIST:ICDAR.



(b) MNIST$_5k$:TiCC$_a$.



(c) MNIST$_{5k}$:TiCC$_d$.



(d) USPS:MADBASE.

Figure 6.2: Number of complete-models/feature detectors in the adaptive part and in supplementary part of aTPL in four scenarios.

scenarios good performance can be achieved when the number of supplementary detectors is small. It also shows that when a large number of source domain representation is reused, it will be harder to adapt. In the next section, we will simulate the effect of unsuitable representation, including the low-scored feature detectors (which are generated by decreasing/flattening the value of the detectors).

6.4.4 Representation Knowledge Adaptation

We now show how the adaptive transferred profile model actually works in terms of transforming the transferred representation so that it is more useful to the target domain. Since the factors $\boldsymbol{\psi}$ are not directly associated with the representation then we only visualise the effect of the factors $\boldsymbol{\lambda}$. We train a sparse RBM on $20,000$ samples of MNIST handwritten digits and extract the feature detectors as shown in Figure 6.3. After that, we corrupt the detectors in three ways. At first, we randomly flatten the value of the feature detectors (by lowering their absolute value) to make them have lower L1 norm scores, in the second we flip the sign of the feature detector, and finally we combine both of these operations. We then transferred these corrupted representations to train the adaptive transferred profile model on another 2000 samples, also from MNIST. We show the visualisation of the representation after the learning.

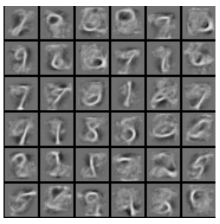
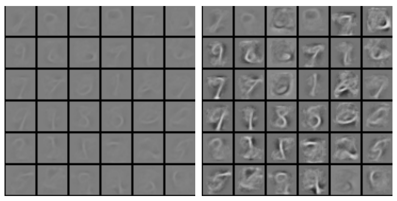
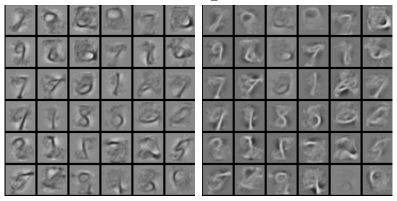


Figure 6.3: Representations (or feature detectors) from sparse RBMs learned on 20,000 MNIST samples.

In each sub-figure in Figure 6.4, the left picture shows the corrupted feature detectors, e.g flattened, flipped, and combined. The right picture shows the detectors which have been transformed after applying Algorithm 8. One can see that almost all the corrupted feature detectors are converted back to the original forms which is more useful to the target domain.

Finally, we show how our approach can deal with problem of negative transfer. We use the corrupted versions of the representation learned from 20,000 MNIST samples in the previous experiment and reuse it for writer recognition with the target dataset extracted from MADBASE. The data consist of 300 training samples from 10 different writers. In this experiment, in order to test the negative transfer


(a) Flattened representation.


(b) Flipped representation.

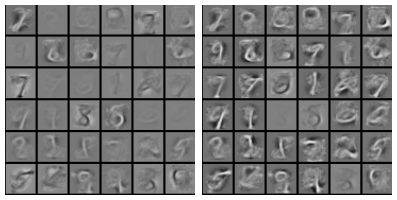

(c) Flattened and flipped representation.

Figure 6.4: The corrupted representations and their transform after applying Algorithm 8 on 2000 MNIST test samples. For each sub-figure, the left picture shows the corrupted representation and the right picture show the representation after the learning.

with features from corrupted source domain, we add Gaussian noise with standard deviation 0.2 to the MNIST data. The representation learned from this corrupted data will be transferred to the target domain.

	Flattened	Flipped	Combined	Corrupted data
SVM	39.43			
RBM	38.92			
RBM STL	10.29	31.14	28.57	40.29
RBM MIX	36.74	34.92	36.57	40.46
aTPL($\beta = 0$)	40.29	37.67	38.79	**42.07**
aTPL($\beta > 0$)	**41.31**$_{\beta=0.5}$	**39.75**$_{\beta=0.01}$	**40.79**$_{\beta=0.01}$	41.43$_{\beta=0.01}$

Table 6.2: Negative transfer with corrupted representation from the source domain, trained on 20,000 MNIST samples. The representations are flattened, flipped, and combined of both effects. The target domain is the MADBASE data set consisting of Hindi handwritten digits from 10 writers. The bold numbers indicate that the improvement is statistically significant.

The results in Table 6.2 show that the corrupted representations severely harm the self-taught learning. In contrast, our approach is capable of adaptively transforming the corrupted representation such that it is still useful for transfer learning.

Furthermore, we observed that in the cases of having many incompatible features transferred, the combination of both direct adaptation factors and indirect adaptation factors, i.e $\beta > 0$, results in a better performance.

We also apply this transfer learning framework to sentiment analysis domain. The preliminary results are shown in Appendix F.

6.5 Summary

This chapter studied further the use of *complete-models* as independent representations for knowledge transfer. The source knowledge has been learned from unlabelled data that will be transformed to adapt with the target domains. In order to obtain the adaptation, an idea of transferred-profile likelihood has been proposed. The transferred likelihood learning estimates subsets of parameters while profiling out other parameters (profile likelihood) by assuming that the other parameters are already estimated in the other domain. Applying this to representation reuse with RBMs, the model then learns the adaptation factors for the transferred representations while estimating new representations in the target domain. We performed intensive experiments to study the effectiveness of our approach. The results showed that the adaptive transferred profile model offers an advantage over self-taught learning and also the combination between self-taught learning and feature learning in the target domain, with better accuracy and more compactness.

Chapter 7

Conclusion and Future Work

7.1 Summary

Unsupervised models such as restricted Boltzmann machines and deep belief networks can learn useful patterns for recognition tasks in wide range of domains. In addition, these patterns have been shown to capture different levels of domain representations. For example, visualisation of the patterns learned from the handwritten image domain indicates that low level patterns represent curves and edges while the higher level patterns represent more concrete shapes. This interesting characteristic of unsupervised learning intrigues a question of "whether symbolic knowledge can also be represented by these patterns?". This thesis studied the decomposition of representations in RBMs/DBNs into symbolic rules to build a learning, knowledge extraction and sharing system.

Chapter 1 has presented the motivation of the work and discussed the idea of representation decomposition to support learning, extraction, and sharing of knowledge.

In **Chapter 2** we revised the theory and applications of energy-based unsupervised learning models. Hopfield networks, Boltzmann machines, restricted Boltzmann machines and deep belief networks have been studied. After that we discussed the promising of such models in knowledge extraction, neural-symbolic integration and transfer learning. The focus of this thesis, as mentioned in this chapter, is on unsupervised layer-wise approach to understand the effectiveness of modular learning and reasoning.

Based on the work of Penalty Logic on symbolic representation and reasoning in energy-based connectionist systems **Chapter 3** studied the relation between propositional logic and restricted Boltzmann machines, and subsequently deep belief networks. This chapter showed that it is possible to present a propositional well-formed formula in an RBM where symbolic reasoning corresponds to energy minimisation. In order to present a propositional formula in a DBN we introduced the idea of confidence rules and proposed the use of confidence values to constrain the minimum energy to preferred assignments and the maximum energy to non-preferred assignments. It can be seen that confidence rules act as an intermediate model between interpretable logical programs and a hierarchical connectionist systems.

With confidence rules have been defined, **Chapter 4** discussed the idea of using them for knowledge extraction and encoding. First, an extraction algorithm has been proposed. For an RBM, the representation is decomposed into a set of feature detectors which is subsequently converted into confidence rules. Second, examples of how confidence rules can be interpreted and what knowledge can be obtained from a domain through the extraction have been shown. Third, "partial-models" were extended from confidence rules. Different from symbolic reasoning in confidence rules,"partial-models" perform hierarchical inference as that in RBMs instead of satisfiability as in symbolic logic programs. The experiments showed that in some cases partial-models can achieve similar performance as RBMs/DBNs while they are more compact and also more symbolic related. Finally, we inverted the extraction process to integrate prior symbolic knowledge into a deep models. The experiments on DNA and MNIST dataset showed that this deep neural-symbolic integration system can take advantage from the given knowledge and achieve better performance in unsupervised layer-wise learning.

Even though integration of background knowledge from a domain can help improve the learning in this domain, domain knowledge is not easy to obtain, especially knowledge for classification tasks. An alternative is to reuse the knowledge that has been learned from a domain by transferring it onto another domain.

Chapter 5 proposed an efficient algorithm to compute the confidence values and use them to rank the representations. These representations have been seen as a set of complete-models which can be reused to transfer to another domain. Experiments showed that a subset of complete-models with highest confidence values can capture the majority of RBMs, and transferring them to another domain can also help improve the learning of predictive features.

The effectiveness of transferred knowledge depends on how well it is suitable for the learning in the target domain. In order to extend the reusability of the extracted knowledge **Chapter 6** proposed an algorithm to adapt the transferred knowledge for better adaptation with the target domain.

7.2 Limitations of the work

This thesis showed that the decomposition of RBMs/DBNs into logical related components (confidence rules) provides an effective means to build a system that incorporates learning, extraction and sharing of knowledge. Although our decomposition approach using Euclidean distance showed the capability of extracting useful knowledge efficiently, we have not studied whether this method is optimal or not. To the best of our knowledge, knowledge extraction from DBNs is new and has not been investigated thoroughly. This makes the comparison to other methods difficult. As we presented in **Chapter 4**, minimising Euclidean distance for rule extraction is heuristic which does not explicitly guarantee the key property of confidence rules, i.e. setting the higher total confidence values for the preferred assignments and the lower ones for the non-preferred assignments. This property, however, can be seen through the empirical examples of XOR and Car Evaluation in §4.1.2 which suggest that the property can be satisfied to a certain extent.

Furthermore, the problem of performance loss during the extraction is a critical issue which should be addressed systematically. As we showed in **Chapter 4**, the loss seems trivial in simple and small domains such as XOR and DNA but with a complex domain as MNIST it is more severe. Theoretical study of the relation between the loss and the complexity of the domain should be considered, for example a mathematical expression to show the correlation between the complexity and the expectation of the loss. However, due to the time constraints we leave it here as an open question and we encourage anyone, who is interested in this topic, to collaborate in investigating further the theoretical concept of the extraction. Intuitively, the loss may be caused by the conversion from real-valued vectors of the weight matrix to discrete vectors representing the rules. While the weights are learned directly from the data by assigning lower energy to the training samples, the extracted rules, however, do not guarantee this assignment. This also raises the question, "Can learning rules directly from data improve the problem of performance loss?". We will discuss this further as a direction for future work in the next section.

The idea of using confidence rules as low-cost representation of RBMs attracts considerable attention from researchers in related fields as it provides an effective means to implement representation learning at hardware level. Some are interested in extending it to larger datasets and also deeper models such as DBNs and SAEs, which has not been done in this thesis. We also believe that this would be beneficial to real-world applications, i.e design of deep network chips, however in this thesis we did not study the low-cost representation further because our focus is on knowledge extraction and sharing.

The transfer learning algorithms we proposed in this thesis have shown promising results in a number of datasets. However, extended experiments on larger data such as natural images with convolutional DBNs and convolutional NNs would provide more understanding of their practical use. Moreover, even though the comparison between transferring representations and transferring data has been conducted in the sentiment analysis experiment in Appendix F, more experiment on other datasets would provide more convincing results. We leave this for the future work on knowledge extraction for transfer learning using multimodal deep networks.

Our main interest is to obtain better rules through either extraction or learning. We expect that confidence rules, with their generative structure, can be useful for knowledge representation, effective symbolic inference and knowledge discovery. In the next section we present several directions for future work.

7.3 Recommendation for Future work

Although the results presented in this thesis have demonstrated that symbolic knowledge can be represented in a hierarchical unsupervised setting, this study is in its beginning and can be further developed in a number of ways:

7.3.1 Extension of Confidence Rules

Learning Confidence Rules

In **Chapter 3** confidence rules have shown their ability of capturing the majority of RBMs/DBNs. These hybrid rules were obtained via a knowledge extraction method from learned models by minimising the distance between the representations and the vectors representing the rules. One can see this as an indirect way to build a symbolic-like program which relies heavily on the learning capability of the models. In future work, this reliance can be avoided by an algorithm that learns confidence rules directly from the data.

This can be considered as learning a compact representation of deep networks which, if successful, will help port deep learning applications to limited memory devices such as mobile phones and tablets. Following a two-phase approach as in other deep learning techniques, it should start with unsupervised learning of a set of rules as:

$$\begin{aligned} &\text{Minimise} \quad \sum_m \text{distance}(\mathbf{x}^{(m)}, \hat{\mathbf{x}}^{(m)}; R) \\ &\text{with} \quad \hat{\mathbf{x}}^{(m)} \leftarrow f(\mathbf{h}^{(m)}, N_R) \text{ and } \hat{\mathbf{h}}^{(m)} \leftarrow g(\mathbf{x}^{(m)}, N_R) \end{aligned} \tag{7.1}$$

where N_R is an unsupervised model to present a set of confidence rules R, and $\mathbf{x}^{(m)}$ is a sample in the training dataset. This is similar to learning a discrete Auto Encoder or RBM where each weights w_{ij} will have three possible values $-c_j, 0, c_j$ with $c_j > 0$ for all j. For that, the continuous-discrete optimisation techniques should be employed to the learn optimal positive/negative combinations of literals. After the unsupervised pre-training, supervised tuning can be applied.

Deep Logic

Confidence rules can be used for symbolic reasoning with hypotheses satisfiability as discussed in **Chapter 3**. If we remove the confidence values and only keep the symbolic part of confidence rules we will have a hierarchical logic program. But the question is:"*Can we build an effective deep symbolic program for symbolic reasoning*"? This question, when answered, will show how a neural system can be learned to represents a symbolic program. This will be known as *Deep Logic* program which is different from the DBLN in **Chapter 4** in that the inference is purely symbolic. Performing symbolic reasoning on the rules can be seen as a strict variant of INF_PARTIAL. This inference rule means the proposition h holds if and only if there are no missing propositions in the premises AND all propositions in the rules must match the premises.

The first objective of this idea is to learn symbolic representations using unsupervised methods. This is similar to the idea of Boolean matrix factorisation [88] in which the symbolic relations can be represented by Boolean operators.

$$\begin{aligned} \text{Minimise} \quad & \|\mathbf{X} - \mathbf{WH}\|_2^2 \\ \text{w.r.t} \quad \mathsf{x}_{im} &= [\mathit{false}, \mathit{true}] \quad \text{for all } i, m \\ \mathsf{w}_{ij} &= [\mathit{false}, \mathit{true}] \quad \text{for all } i, j \\ \mathsf{h}_{jm} &= [\mathit{false}, \mathit{true}] \quad \text{for all } j, m \end{aligned} \tag{7.2}$$

However the inference in Boolean matrix factorisation is conducted through an optimisation process. For example, given a Boolean vector $\mathbf{x}$ denoting a set of propositions the hidden proposition $\mathbf{h}$ can be found by minimising $\sum_i(\mathsf{x}_i - \sum_j(\mathsf{w}_{ij}\mathsf{h}_j))^2$. In this case, algebra *sum* and *product* operators behave as Boolean $\vee$ and $\wedge$ operators, i.e *true* + *true* = *true*. This Boolean model is equivalent to a set of disjunctive if-and-only-if rules: $\mathsf{x}_i \leftrightarrow \bigvee_{j,w_{ij}=1} \mathsf{h}_j$. Different from this, in **Chapter 3** and **Chapter 4** we have shown that in order to capture the semantic in a wide range of domains the rules should be in conjunctive if-and-only-if. However, it is interesting to investigate whether it is possible to learn a Boolean unsupervised model such as a Boolean RBM under the similar constraints as in Eq. 7.2. After that a stack of such Boolean RBMs can be seen as a hierarchical logic program.

7.3.2 Deep Relational Networks

Deep learning has been successfully applied to conventional data where each data sample is treated as a vector. The relationships of domain variables however, are not taken into account. In relational domains, the data consists of background knowledge $\mathbb{B}$ and examples $\mathbb{E}$ in the the form of first-order ground facts or rules. In this research area, the questions of interest are:

- Learning: How to induce hypothesised logic program/relational model from background knowledge and examples?
- Reasoning: Given a relational model how to give conclusions from new coming facts?

In order to deal with uncertainty, probabilistic models have been proposed (see [43] for a list of models). The main idea of statistical relational AI is to encode relational rules into a probabilistic model to take advantage of its reasoning scheme. This model can also be used to learn hypothesis for the domain by inducing model's structure from background knowledge and examples. We will show that, using the theory developed in this thesis we can build a deep relational model for both learning and reasoning.

Let us consider an example with three predicates $Mother\_of(z,x)$, $Father\_of(y,x)$, $Husband\_of(y,z)$ which present the relations between people, e.g $Mother\_of(Mary, John) = true$ means Mary is mother of John. For each assignment of a predicate we can convert it to a propositional conjunction, for example $Mother\_of(Mary, John)$ becomes $\mathsf{mother} \wedge \mathsf{z}_{\mathsf{Mary}} \wedge \mathsf{x}_{\mathsf{John}}$. Here mother is proposition indicating whether the predicate is $Mother\_of$ or not, $\mathsf{z}_{\mathsf{Mary}}$ and $\mathsf{x}_{\mathsf{John}}$ are propositions of assigning $Mary$, $John$ to variable z and x respectively. As discussed in **Chapter 3** this relation can be presented as a confidence rule $c_j : \mathsf{h}_j \leftrightarrow \mathsf{mother} \wedge \mathsf{z}_{\mathsf{Mary}} \wedge \mathsf{x}_{\mathsf{John}}$. Furthermore, if we group the confidence rules constructed from all possible assignments for a predicate we can present exactly this predicate. From these rules we can create an RBM. In order to create a higher level of relationships, we group all hypotheses from a predicate and create a max-pooling layer [80] such that an unit p_k (a predicate unit) is activated only when at least one hypothesis in the predicate is *true*. After that another hidden

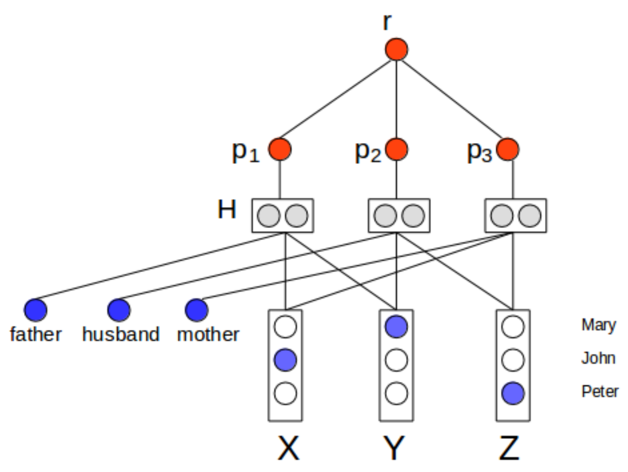


Figure 7.1: Proposed deep relational model.

layer is added to present the higher relations. In Figure 7.1 we show a model that presents three predicates above. We expect that this model may not only be used for prediction but also for knowledge exploration.

7.3.3 Multimodal Learning-Extraction-Sharing

In the current trend of Big Data, multimodality is emerging to be an important and challenging topic in data science. Even though there exist several models proposed for learning from multiple data sources such as visual, text and audio, little or no study has been conducted on extraction and sharing of knowledge in these domains. Our future work will primarily focus on developing a comprehensive system that provides efficient learning, extraction and sharing of knowledge from multimodal data. The objectives of the research are: (i) *To develop a representation learning model that captures spatial/temporal aspects of the mutimodal data?*, (ii)*To understand whether interpretable knowledge can be obtained from multimodality through extraction from this model*, (iii) *To investigate whether the learned/extracted knowledge can be effectively reused to improve the learning in other (unimodal/multimodal) domains.*

In order to obtain the first objective we will develop a novel deep learning model that captures the background knowledge of each modality. Current multimodal deep learning approaches tend to treat all modalities similarly [98, 128], while in this proposed model the spatial information of images/video will be learned by using 2D/3D convolution, and the temporal information of text/video/audio will be learned by using recurrence. The effectiveness of the model will be tested on benchmark datasets of sentiment analysis and emotion recognition.

For the second objective we will develop new algorithms to extract knowledge from the multimodal system mentioned above. This study will base on recent research on confidence rules, visual semantic [25] and image description [68]. It will extend modal logic to capture complex possibilities of temporal representation of sequence data. We are also interested in studying how to use modal logic to represent visual semantics. The algorithms will be evaluated in human behaviour analysis.

Finally, to complete the system with the sharing of learned/extracted knowledge we will develop a transfer mechanism to allow knowledge to be reused in other domains. The transfer will be adaptive so that knowledge will be selected/altered to improve the learning in new domains.

Bibliography

[1] Maruan Al-Shedivat, Jim Jing-Yan Wang, Majed Alzahrani, Jianhua Huang, and Xin Gao. Supervised transfer sparse coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

[2] Harith Alani, Sanghee Kim, David Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis, and Nigel R. Shadbolt. Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18(1):14–21, January 2003.

[3] Harith Alani, Sanghee Kim, David Millard, Mark J. Weal, Paul H. Lewis, Wendy Hall, and Nigel R. Shadbolt. Automatic extraction of knowledge from web documents. In *Proceedings of the International Semantic Web Conference - Workshop on Human Language Technology for the Semantic Web abd Web Services*, 2003.

[4] Hazrat Ali, Son Tran, Artur S. dAvila Garcez, and Tillman Weyde. Convolutional data: Towards deep audio learning from big data. In *Proceedings of the 1st UCL Workshop on the Theory of Big Data*, 2010.

[5] James A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(34):197 – 220, 1972.

[6] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, December 1995.

[7] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[8] Artur S. Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, July 1999.

[9] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transaction on Pattern Analysis Machine Intelligence*, 24(4):509–522, April 2002.

[10] Yoshua Bengio. Learning deep architectures for AI. *Foundation and Trends in Machine Learning*, 2(1):1–127, January 2009.

[11] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160. MIT Press, 2007.

[12] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. *Large Scale Kernel Machines*, 2007.

[13] Mathias Berglund, Tapani Raiko, and Kyunghyun Cho. Measuring the usefulness of hidden units in boltzmann machines with mutual information. *Neural Networks*, 64(0):12 – 18, 2015. Special Issue on Deep Learning of Representations.

[14] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

[15] Edwin Bonilla, Kian Chai, and Christopher Williams. Multi-task Gaussian process prediction. In *Advances in Neural Information Processing Systems*. 2008.

[16] Rafael V. Borges, Artur S. d'Avila Garcez, and Luís C. Lamb. Learning and representing temporal knowledge in recurrent networks. *IEEE Transactions on Neural Networks*, 22(12):2409–2421, 2011.

[17] Léon Bottou. From machine learning to machine reasoning: an essay. *Machine Learning*, 94:133–149, January 2014.

[18] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[19] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[20] John Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing*, pages 227–236, 1990.

[21] Gertjan J. Burghouts, Henri Bouma, Richard den Hollander, Bas van den Broek, and Klamer Schutte. Recognition of 48 human behaviors from video. In *Proceedings of the International Symposium on Optronics in Defence and Security*, 2012.

[22] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

[23] George Casella and Edward I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[24] Bo Chen, Jo-Anne Ting, Benjamin Marlin, and Nando de Freitas. Deep learning of invariant spatio-temporal features from video. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[25] Xinlei Chen, Abhinav Shrivastava, and Abhinav Gupta. Neil: Extracting visual knowledge from web data. In *Proceedings of the IEEE International Conference on Computer Vision*, December 2013.

[26] Srikanth Cherla, Tillman Weyde, Artur S d'Avila Garcez, and Marcus Pearce. A Distributed Model for Multiple-Viewpoint Melodic Prediction. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 15–20, 2013.

[27] Mark W. Craven and Jude W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 73–80. Morgan Kaufmann, 1993.

[28] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Co-clustering based classification for out-of-domain documents. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 210–219, 2007.

[29] Wenyuan Dai, Gui-rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 540–545, 2007.

[30] Wenyuan Dai, Qiang Yang, Gui-rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the International Conference in Machine Learning*, 2007.

[31] Artur d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1–2):155–207, 2001.

[32] Artur d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Perspectives in Neural Computing. Springer, 2002.

[33] Artur d'Avila Garcez, Luis C Lamb, and Dov M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer, 2009.

[34] Jesse Davis and Pedro Domingos. Deep transfer via second-order markov logic. In *Proceedings of the International Conference on Machine Learning*, Proceedings of the International Conference on Machine Learning, page 217224, New York, NY, USA, 2009.

[35] Anthony Davison. *Statistical Models*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 1 edition, Aug 2003.

[36] Leo de Penning, Artur S d'Avila Garcez, and John-Jules Ch Meyer. Dreaming machines: On multimodal fusion and information retrieval using neural-symbolic cognitive agents. In *Proceedings of the Imperial College Computing Student Workshop*, 2013.

[37] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine Learning*, pages 264–271, New York, NY, USA, 2008. ACM.

[38] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, March 2010.

[39] Theodoros Evgeniou and Massimiliano Pontil. Regularized multitask learning. In *Proceedings of the tenth International Conference on Knowledge Discovery and Data Mining*, page 109117, 2004.

[40] Emile Fiesler and Russell Beale. *Handbook of Neural Computation*. Oxford University Press, 1997.

[41] Dov M. Gabbay. Labelled deductive systems: a position paper. In J. Oikkonen and J. Vnnen, editors, *Logic Colloquium '90: ASL Summer Meeting in Helsinki*, Lecture Notes in Logic, pages 66–68. Springer-Verlag, Berlin, 1993.

[42] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, page 283291, 2008.

[43] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[44] Walter Gilks, Sylvia. Richardson, and David Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC, 1995.

[45] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[46] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the International Conference on Machine Learning*, pages 513–520, 2011.

[47] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[48] Barbara Hammer and Pascal Hitzler, editors. *Perspectives of Neural-Symbolic Integration*. Springer, 2007.

[49] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[50] Federico Heras, Javier Larrosa, and Albert Oliveras. MINIMAXSAT: An Efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[51] Geoffrey Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1-3):185–234, September 1989.

[52] Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, August 2002.

[53] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 599–619. 2012.

[54] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[55] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[56] Geoffrey Hinton and Ruslan Salakhutdinov. A better way to pretrain deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2447–2455. 2012.

[57] Geoffrey Hinton and Terrence Sejnowski. Learning and relearning in boltzmann machines. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 282–317, 1986.

[58] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[59] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Diploma thesis, 1993.

[60] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1-2):177–196, 2001.

[61] John Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[62] Derek Hao Hu and Qiang Yang. Transfer learning for activity recognition via sensor mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1962–1967, 2011.

[63] Jiayuan Huang, Alexander Smola, Arthur Gretton, Karsten Borgwardt, and Bernhard Schlkopf. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19*, pages 601–608, 2007.

[64] Alekseĭ GrigorŁevich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1965.

[65] Aleksey Grigorievitch Ivakhnenko, Valentin Grigorievitch Lapa, and Robert N McDonough. *Cybernetics and forecasting techniques*. American Elsevier, NY, 1967.

[66] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[67] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, November 1999.

[68] Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306, 2014.

[69] Solomon Kullback and Richard Leibler. On information and sufficiency. *The Annals of Mathematics and Statistics*, 22(1):79–86, 1951.

[70] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, pages 536–543, 2008.

[71] Richard Lassaigne and Michel de Rougemont. Propositional logic. In *Logic and Complexity*. Springer-Verlag London, London, UK, 2004.

[72] Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *Proceedings of the International Conference on Machine Learning*, page 65, 2004.

[73] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

[74] Quoc Le, Will Y. Zou, Serena Y. Yeung, and Andrew Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3361–3368, 2011.

[75] Yann LeCun, Bernhard Boser, John. Denker, D. Henderson, Richard. Howard, Wayne Hubbard, and Lawrence Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, December 1989.

[76] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19*, pages 801–808. MIT Press, Cambridge, MA, 2007.

[77] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*, 2007.

[78] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the International Conference on Machine Learning*, pages 609–616, 2009.

[79] Honglak Lee, Yan Largman, Peter Pham, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems*, pages 1096–1104. 2009.

[80] Honglak Lee, Peter T. Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems*, pages 1096–1104, 2009.

[81] Su-in Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller. Learning a meta-level prior for feature relevance from multiple related tasks. In *Proceedings of International Conference on Machine Learning*, pages 489–496, 2007.

[82] Zhaoping Li and Peter Dayan. Computational differences between asymmetrical and symmetrical networks. In *Advances in Neural Information Processing Systems*, pages 274–280, 1998.

[83] Percy Liang and Michael I. Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proceedings of the 25th International Conference on Machine Learning*, Proceedings of the International Conference on Machine Learning, pages 584–591, New York, NY, USA, 2008. ACM.

[84] Zhe Lin, Zhuolin Jiang, and Larry S. Davis. Recognizing actions by shape-motion prototype trees. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 444–451. IEEE, 2009.

[85] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[86] Christopher D Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[87] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Neurocomputing: Foundations of Research*, pages 15–27, 1988.

[88] Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 51–59, 2011.

[89] Lilyana Mihalkova, Tuyen Huynh, and Raymond J. Mooney. Mapping and revising markov logic networks for transfer learning. In *Proceedings of the AAAI Conference on Artificial intelligence*, pages 608–614, 2007.

[90] Lilyana Mihalkova and Raymond J. Mooney. Transfer learning from minimal target data by mapping across relational domains. In *Proceedings of the International Jont Conference on Artifical intelligence*, pages 1163–1168, 2009.

[91] Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):14–22, 2012.

[92] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[93] Vinod Nair and Geoffrey Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning*, 2010.

[94] Radford Neal. Probabilistic inference using markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[95] Radford Neal and Geoffrey Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. MIT Press, Cambridge, MA, USA, 1999.

[96] Andrew Y Ng and Michael I Jordan. On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848, 2001.

[97] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems*, pages 841–848. MIT Press, 2002.

[98] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *Proceedings of the International Conference on Machine Learning*, Bellevue, USA, June 2011.

[99] Juan Carlos Niebles, Hongcheng Wang, and Li Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision*, 79(3):299–318, 2008.

[100] Bruno Olshausen and David Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[101] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[102] Marcus Pearce and Geraint Wiggins. Improved Methods for Statistical Modelling of Monophonic Music. *Journal of New Music Research*, 33(4):367–385, 2004.

[103] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000.

[104] Barak A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Compution*, 1(2):263–269, June 1989.

[105] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[106] Leo de Penning, Artur S. d'Avila Garcez, Lus C. Lamb, and John-Jules Ch Meyer. A neural-symbolic cognitive agent for online learning and reasoning. In *Proceedings of the International Joint Conference on Artificial Inteligence*, pages 1653–1658, 2011.

[107] Gadi Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 525–530, 1991.

[108] Gadi Pinkas. Symmetric neural networks and propositional logic satisfiability. *Neural Computation*, 3(2):282–291, June 1991.

[109] Gadi Pinkas. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. *Artificial Intelligence*, 77(2):203–247, September 1995.

[110] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, Proceedings of the International Conference on Machine Learning, page 759766, New York, NY, USA, 2007. ACM.

[111] Marc'Aurelio Ranzato, Y.-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems*, 2007.

[112] Mauricio Resende, Leonidas Pitsoulis, and Panos Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In *Satisfiability problem: Theory and Applications*, volume 35, pages 393–405. 1997.

[113] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, February 2006.

[114] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[115] David Rumelhart, Geoffrey. Hinton, and Ronald Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[116] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. In *Neurocomputing: Foundations of Research*, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[117] Stuart Russell and Peter Norvig. Knowledge, reasoning, and planning. In *Artificial Intelligent: A Modern Approach*. Pearson Education, 2003.

[118] Stuart Russell and Peter Norvig. Logical agents. In *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[119] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *AISTATS*, pages 448–455, 2009.

[120] Ruslan Salakhutdinov, Joshua B. Tenenbaum, and Antonio Torralba. Learning with hierarchical-deep models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1958–1971, 2013.

[121] Helmut Schaffrath and D Huron. The Essen Folksong Collection in the Humdrum Kern Format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*, 1995.

[122] Juergen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[123] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ICPR '04, pages 32–36, Washington, DC, USA, 2004. IEEE Computer Society.

[124] Paul Smolensky. Connectionist AI, symbolic AI, and the brain. *Artificial Intelligence Review*, 1(2):95–109, 1987.

[125] Tran Son, Wolff Daniel, Tillman Weyde, and Artur dAvila Garcez. Feature preprocess- ing with restricted boltzmann machine for music similarity learning. In *Audio Engineering Society 53rd conference on Semantic Audio*, 2014.

[126] Son Tran and Artur Garcez. Logic extraction from deep belief networks. In *ICML 2012 Representation Learning Workshop*, Edinburgh, July 2012.

[127] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[128] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. *Journal of Machine Learning Research*, 15:2949–2980, 2014.

[129] Sebastian Stober and Andreas Nürnberger. Similarity adaptation in an exploratory retrieval scenario. In *Proceedings of Adaptive Multimedia Retrieval*, Linz, Austria, Aug 2010.

[130] Sebastian Stober and Andreas Nürnberger. An experimental comparison of similarity adaptation approaches. In *Proceedings of the Adaptive Multimedia Retrieval*, Barcelona, Spain, Jul 2011.

[131] Ilya Sutskever, Geoffrey Hinton, and Graham Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 1601–1608. MIT Press, 2008.

[132] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

[133] Matthew Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:16331685, December 2009.

[134] Tijmen Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *Proceedings of the International Conference on Machine Learning*, pages 1064–1071. ACM New York, NY, USA, 2008.

[135] Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Transfer learning via advice taking. In *Advances in Machine Learning*, pages 147–170. Springer, 2010.

[136] Geoffrey G. Towell and Jude W. Shavlik. The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, pages 71–101, 1993.

[137] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.

[138] Son Tran and Artur d'Avila Garcez. Knowledge extraction from deep belief networks for images. In *IJCAI-2013 Workshop on Neural-Symbolic Learning and Reasoning*, 2013.

[139] Yang Wang and Greg Mori. Human action recognition by semilatent topic models. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 31(10):1762–1774, October 2009.

[140] Daniel Wolff. *Spot the Odd Song Out: Similarity Model Adaptation and Analysis using Relative Human Ratings*. PhD thesis, City University London, Northampton Square, London, UK, August 2014.

[141] Daniel Wolff, Sebastian Stober, Andreas Nürnberger, and Tillman Weyde. A systematic comparison of music similarity adaptation approaches. In *Proceedings of the International Society for Music Information Retrieval Conference*, 2012.

[142] Jie Xu, Getian Ye, Yang Wang, Gunawan Herman, Bang Zhang, and Jun Yang. Incremental EM for probabilistic latent semantic analysis on human action recognition. In *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 55–60, 2009.

[143] Geoffrey Hinton Yann LeCun, Yoshua Bengio. Deep learning. *Nature*, 521:436–444, 2015.

[144] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[145] Steven J Young and Sj Young. The htk hidden markov model toolkit: Design and philosophy. *Entropic Cambridge Research Laboratory, Ltd*, 2:2–44, 1994.

[146] Jianguo Zhang and Shaogang Gong. Action categorization by structural probabilistic latent semantic analysis. *Computer Vision and Image Understanding*, 114(8):857–864, August 2010.

[147] Blaz Zupan, Marko Bohanec, Ivan Bratko, and Janez Demsar. Machine learning by function decomposition. In *Proceedings of the International Conference on Machine Learning*, pages 421–429, 1997.

Appendix A

Applications of Representation/Deep Learning

A.1 Music Similarity

Similarity measurement is the key component in search engines, recommendation systems and games. This section applies representation/deep learning to improve the performance of a similarity model which has been used in the *Spot the odd song out* game developed by Daniel Wolff [140]. The game shows three songs to a player and asks him/her to identify the song that is different from the other two. To support this game a similarity model should be used to find out *given a song* $\mathbf{x}$ *which one of the other songs* $\mathbf{y}$, $\mathbf{z}$ *is more similar to it*. The distance vector between two songs is defined as:

$$\text{dist}(x, y) = (\mathbf{x} - \mathbf{y}) \circ (\mathbf{x} - \mathbf{y}) \tag{A.1}$$

The similarity between two songs, say $\mathbf{x} \in \mathbb{R}^I$ and $\mathbf{y} \in R^I$, is measured using the weighted euclidean metric as:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \sum_i^I a_i \text{dist}(\mathbf{x}, \mathbf{y})_i \tag{A.2}$$

where a_i is the weight parameter. For a triplet $\{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$ of the training set $\mathcal{D}$ the similarity relation that $\mathbf{x}$ is more similar to $\mathbf{y}$ than to $\mathbf{z}$ can be seen as $\text{sim}(\mathbf{x}, \mathbf{y}) <$

sim($\mathbf{x}, \mathbf{z}$). One may want to learn the parameter vector **a** that:

$$\text{Maximise } \Gamma = C \sum_{\{\mathbf{x},\mathbf{y},\mathbf{z}\} \in \mathcal{D}} \mathbf{a}^\top(\text{dist}(\mathbf{x}, \mathbf{z}) - \text{dist}(\mathbf{x}, \mathbf{y})) - \frac{1}{2}||\mathbf{a}||^2 \tag{A.3}$$

where C is a penalty value. This optimisation problem can be solved by using gradient ascent or a soft-margin approach using SVM, as presented in [130, 141]. Applying representation learning to this model, the original features **x**, **y**, **z** will be replaced by the latent features learned from a binary RBM, for example $\mathbf{h}_x$, $\mathbf{h}_y$, $\mathbf{h}_z$.

	Features		
Appr.	Original	PCA	RBM
GRAD	70.47 / 71.68	70.54 / 70.52	73.14 / 73.28
SVM	71.20 / 83.54	70.17 / 75.29	72.18 / 80.17

Table A.1: Comparison of original features and those with PCA and RBM pre-processing in terms of similarity prediction accuracy. Test and training set results are listed as percentages of correctly predicted similarity constraints for the configurations with the best training success. The SVM original values are taken from [141].

Table A.1 shows the performance of different feature extraction techniques. For completeness PCA is also included. The best result of the model trained by gradient approach within 20 runs is reported for each RBM parameterisation. Unfortunately, the SVM is much more computationally expensive, then the results of single runs are displayed for this approach. The results for gradient ascent (GRAD) on original features are comparable to those published in [141]. One should note that this gradient approach is similar to that in [129, 141], with the difference is in the latter the weights a_i are constrained to $\sum a_i = 1$. As shown in Table A.1, the PCA transformation of music data seems not to work well with the SVM training, these features are slightly worse than in the original features, while using the gradient approach shows little improvement. In contrast, improvement can be seen in all approaches for the RBM features, with gradient ascent has the best test results, improving by 2.67% over the original features, while SVM gains 0.92%.

A.2 Action Recognition

Human action recognition is considered to be a fundamental topic in computer vision research, with numerous applications in surveillance and retrieval systems [21, 66].Typically, action recognition systems model video recordings as collections of visual words, which are estimated using hand-crafted features. Extracting features from each frame image to build code-words has been proved an efficient and useful approach [146, 9, 60].

In this application, we employ Gaussian RBMs to efficiently learn spatio-temporal features using a difference measure between frames in a video sequence, called motion-difference. A motion-difference (MD) is the subtraction of $I_{t+\kappa}$ by I_t, two images at positions $t + \kappa$ and t in a video sequence respectively, κ is frame distance. As result, in motion-difference the negative pixels show the part of an actor which only appears in the previous frame (t) while the positive pixels show the portion only in the future frame ($t + \kappa$). Motion-difference removes the common shapes and background images that should not be relevant for action learning and recognition, and highlights the movement patterns in space, making it easier to learn the actions from such saliency maps using a simple classifier. This application also employs representation learning to learn the movement patterns from motion different features, using Gaussian RBMs. The latent features from RBMs then will be converted into motion-difference words using K Nearest Neighbour.

We evaluate the visual words using the Naive Bayes and probabilistic Latent Semantic Analysis (pLSA) classifiers. We report our results on two datasets: Weizmann and KTH , along with results from other approaches regarding to different classes of features such as shape descriptor (SD), motion descriptor (MF), hand-crafted spatio-temporal (HST), and learned spatio-temporal (LST) descriptor. In order to make a fair comparison, here we emphasise the approaches that use similar classification models such as Naive Bayes and pLSA or their variants. For completeness, we also include the recent approaches which achieve state-of-the-art performance. Significance comparisons between the approaches is not possible since each employed different reprocessing and classification techniques. In addition, each approach adopts different method such as split or leave-one-out (l-o-o) for experimental evaluation.

Method	Eval	Recog.rate(%)
MD + NB	split	98.81
MD + pLSA	split	98.77
SD + pLSA [146]	split	92.3
SD + s-pLSA [146]	split	93.00
ST + pLSA [99]	l-o-o	90.00
MF + SVM [139]	l-o-o	98.80
SD & MF[84]	l-o-o	100.0

Table A.2: Performance on Weizmann dataset. The results of [146, 99, 139, 84] are copied from the original papers

Approach	Eval	Recog.rate(%)
MD + NB	split	85.65
MD + pLSA	split	88.89
HST + pLSA [99]	l-o-o	83.33
MF + SVM [139]	l-o-o	83.31
HST + SVM [123]	split	71.72
HST + iEM+pLSA [142]	l-o-o	82.33
LST + SVM [24]	split	86.6
SD + S-LDA [139]	l-o-o	91.20
LST + SVM [74]	split	93.9

Table A.3: Performance on KTH dataset. Results of [99, 139, 123, 142, 24, 74] are copied from the original papers

Table A.2 and Table A.3 show that using RBMs for learning motion-difference features can achieve good performance among state-of-the-art approaches.

A.3 Speaker Recognition

For speaker recognition, the same pipeline as in §A.2 is employed. In order to build the vocabulary of the audio words we start with converting a audio script into spectrogram. We consider the spectrogram of a script as a 2-D matrix with time × frequency dimensions. The number of frequencies is varied, depending on different types of audio coding. We reduce the dimensions of the data for further processing steps by applying PCA to linearly transform the frequencies to lower dimension space. After that we learn the latent features of the audio data using Deep Belief Networks where each input sample is the PCA-transformed frequencies at a time slot. We then use KNN to build a codebook from the DBN features and quantize the features into audio words. At the end, the audio script will be represented in a bag of words, i.e. a vector of audio word counts. In the experiment, we test different types of features. The MFCC features [145] attempt to eliminate information from speech data that is not relevant for recognition purposes, thus providing input representation of modest size. DBNs on the other hand make use of less-processed input data. Instead, it learns the latent features from the PCA-transformed spectrograms. The advantage of DBNs is it can learn useful representation as we can see from the results in Table A.4. Here, taking the advantage of layer-wise learning in DBNs we also combine different features in different layers. Let us denote DBN-1

Approach	Accuracy
MFCC	88.6
Audio words (DBN-1)	90.40
Hybrid (DBN-1+MFCC)	91.40
Audio words (DBN-2)	72.20
Hybrid (DBN-2+MFCC)	87.00
Audio words (DBN-1 + DBN-2)	90.60
Hybrid (DBN-1 + DBN-2 + MFCC)	92.60

Table A.4: Test set accuracy for speaker classification

and DBN-2 as the features from the first and the second RBMs in deep networks.

The results in Table A.4 show that the audio words built from features in first layer of DBNs (DBN-1) outperform the MFCC features. The classification performance even achieves improvement when we combine the audio words generated from DBN-1 features with the MFCC. However, when one more layer is used in the DBNs the the features are not good enough to build audio words and generalise the classifier. It seems that the DBN-1 features generalise better than the DBN-2 counterpart because the expansion of the feature's dimension in fist layer make it more difficult to learn in the second one. This effect also can be seen when applying the convolutional DBNs on audio data [79], but in this experiment it is more severe. As the results, combining DBN-1 and DBN-2 features does not show any improvement.

A.4 Melody Modelling

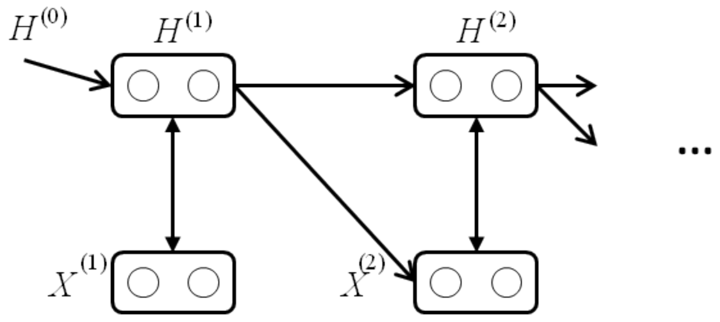
The task we are insterested here is music prediction which is closely related with previous work in language modelling [86]. Let denote a musical event at time t as $s^{(t)}$. A musical event corresponds to the occurrence of a note in a melody. In this application, we want to learn a model that predict a future event given the previous events, i.e $P(s^{(t)})|s^{(1:t-1)})$.

In sequence modelling, the temporal relation of data is normally defined by Markov assumption where the current state only depends on n previous states, i.e $P(s^{(t)})|s^{(1:t-1)}) = P(s^{(t)})|s^{(t-n:t-1)})$. The simplest models go tightly with this assumption is the *n-gram* model. In this model, the conditional distributions are learned to be:

$$P(s^{(t)})|s^{(t-b:t-1)}) = \frac{\text{count}(s^{(t-b:t-1)})}{\text{count}(s^{(t-b:t)})} \tag{A.4}$$

where $\text{count}(s^{(t_x:t_y)})$ is the occurrence of the events $s^{(t_x:t_y)}$ in training scripts and n is context length. Another approach is to employ an algorithmic model such as feed-forward Neural Network (NN) to learn discriminatively the function $f(s^{(t)}); s^{(t-n:t-1)})$. The disadvantages of both *n-gram* and NN are they cannot model the whole sequence of data and the context length must be pre-defined. In order to get rid of the context length, one can unbound the order in the *n-gram* [102] and add recurrent connections to the NN [20].

Another way to model the whole sequence is to use generative model such as the Recurrent Temporal RBMs (RTRBM) [131] to represent the joint distribution $P(s^{(1:T)})$, where T is the length of the sequence. To use RTRBMs for melody modelling, we set each visible layer of a RBMs to be a pitch such that $X^{(t)} = \mathbf{s}^{(t)}$. Here $\mathbf{s}^{(t)}$ is the softmax vector of $s^{(t)}$. Furthermore, for prediction task a discriminative learning seems to perform better than generative learning [96, 70].



Experiment and evaluation was carried out by Srikanth Cherla on a corpus of 8 datasets of monophonic MIDI melodies from the Essen Folk Song Collection[1] [121]. The corpus covers a range of musical styles and was previously used in [102, 26] to evaluate their respective prediction models. It contains folk melodies of 7 different traditions, and chorale melodies

Table A.5 contains the best predictive performance of each of the models considered in the comparison here. The results are averaged across all 8 datasets. Here, the n-gram, FNN, RBM, DRBM, RNN, RTRBM, RTDRBM indicate n-gram

[1]Website: http://kern.ccarh.org/browse?l=essen

Model	Cross Entropy
n-gram	2.878
FNN	2.830
DRBM	2.819
RBM	2.799
RNN	2.778
RTRBM	2.764
RTDRBM	2.741

Table A.5: Table comparing the best predictive performance of the different models in the evaluation. The RTDRBM outperforms the rest.

model,feed-forward Neural Networks, Restricted Boltzmann Machines, discriminative RBM, recurrent Neural Networks, recurrent temporal RBM, and recurrent temporal dicriminative RBM respectively. One will notice the progressive improvement in the best-case performance from the n-gram models, to the non-recurrent and recurrent connectionist models, with the RTDRBM performing better than the rest. A paired t-test carried out over all the 10 resampling sets of each dataset (n = 80) confirmed the significance of the improvement due to the RTDRBM over the RTRBM [$t(79) = 3.65, p < 0.001$] and the RNN [$t(79) = 3.70, p < 0.001$].

Appendix B

Detail of Derivations

B.1 Update of RBMs

The average log-likelihood

$$\ell = \frac{1}{M}\sum_{m} \log P(\mathbf{x}^{(m)}|\theta) \tag{B.1}$$

where M is number of training samples, $\mathbf{x}^{(m)}$ is a sample in the training set, θ is a set of parameters that include W, a, b. For a sample $\mathbf{x}$ (now $p(\mathbf{x}|\theta)$ is replaced by $p(\mathbf{x})$ for shorter notation) we have:

$$p(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z}\sum_{\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h})) \tag{B.2}$$

with the energy function:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{ij} x_i w_{ij} h_j - \sum_{i} a_i x_i - \sum_{j} b_j h_j \tag{B.3}$$

If we take derivation from one sample:

$$\begin{aligned}
\frac{\partial \log p(\mathbf{x})}{\partial\theta} &= \frac{\partial \log(\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h})))}{\partial\theta} - \frac{\partial \ln(Z)}{\partial\theta} \\
&= \frac{1}{\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}))} \sum_{\mathbf{h}} \frac{\partial \exp(-E(\mathbf{x},\mathbf{h}))}{\partial\theta} - \frac{1}{Z}\sum_{\mathbf{x}',\mathbf{h}} \frac{\partial \exp(-E(\mathbf{x}',\mathbf{h}))}{\partial\theta} \\
&= -\frac{1}{\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}))} \sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}))\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta} + \frac{1}{Z}\sum_{\mathbf{x}',\mathbf{h}} \exp(-E(\mathbf{x}',\mathbf{h}))\frac{\partial E(\mathbf{x}',\mathbf{h})}{\partial\theta} \\
&= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{x})\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta} + \sum_{\mathbf{x}',\mathbf{h}} p(\mathbf{h},\mathbf{x}')\frac{\partial E(\mathbf{x}',\mathbf{h})}{\partial\theta}
\end{aligned} \tag{B.4}$$

Let us take the derivation of first term on w_{ij}

$$\begin{aligned}
\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{x})\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial w_{ij}} &= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{x}) x_i h_j \\
&= -\sum_{h_1} \dots \sum_{h_{j'}} \dots \sum_{h_J} \prod_{j'} p(h_{j'}|\mathbf{x}) x_i h_j \\
&= -\sum_{h_1} \dots \sum_{h_{\setminus j}} \dots \sum_{h_J} \prod_{\setminus j} p(h_{\setminus j}|\mathbf{x}) \sum_{h_j} p(h_j|\mathbf{x}) x_i h_j) \\
&= -\sum_{h_1} \dots \sum_{h_{\setminus j}} \dots \sum_{h_J} \prod_{\setminus j} p(h_{\setminus j}|\mathbf{x})(0 + p(h_j = 1|\mathbf{x}) x_i) \\
&= -P(h_j|\mathbf{x}) x_i
\end{aligned} \tag{B.5}$$

For the second term

$$\begin{aligned}
\sum_{\mathbf{x}',\mathbf{h}} p(\mathbf{h},\mathbf{x}')\frac{\partial E(\mathbf{x}',\mathbf{h})}{\partial w_{ij}} &= -\sum_{\mathbf{x}'}(\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{x}') x'_i h_j) p(\mathbf{x}') \\
&= -\sum_{\mathbf{x}'}(P(h_j|\mathbf{x}') x'_i) p(\mathbf{x}')
\end{aligned} \tag{B.6}$$

Note that $P(h_j|\mathbf{x})$ is the probability of $h_j = 1$ given $\mathbf{x}$. Now apply Eq. B.5 and Eq. B.6 to Eq. B.4 for a parameter w_{ij} we have:

$$\frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} = P(h_j|\mathbf{x}) x_i - \sum_{\mathbf{x}'} P(h_j|\mathbf{x}') x'_i p(\mathbf{x}') \tag{B.7}$$

Applying this to all samples to get the derivation in Eq. B.1 as:

$$\frac{\partial \ell}{\partial w_{ij}} = \frac{1}{M}\sum_{m} P(h_j|\mathbf{x}^{(m)}) x_i^{(m)} - \frac{M}{M}\sum_{\mathbf{x}} P(h_j|\mathbf{x}) x_i p(\mathbf{x}) \tag{B.8}$$

Now we can the update of parameter w_{ij} as:

$$\nabla w_{ij} = \langle P(h_j|\mathbf{x})x_i\rangle_0 - \langle P(h_j|\mathbf{x})x_i\rangle_\infty \tag{B.9}$$

here $\langle\rangle_0$ means (empirical) expectation over the data distribution and $\langle\rangle_\infty$ means expectation over model distribution. We can use the same process to calculate the updates for other parameters.

B.2 Energy function for XOR

The propositional calculus Penalty Logic is defined in [109] as: $E = 1 - H_{\mathsf{p}}$, with

$$H_{\mathsf{p}} = \begin{cases} x & \text{if } \mathsf{p} \text{ is a positive atom literal x} \\ 1 - x & \text{if } \mathsf{p} \text{ is a negative atom literal}\neg\text{x} \\ H_{\mathsf{p}'} \times H_{\mathsf{p}''} & \text{if } \mathsf{p} = \mathsf{p}' \wedge \mathsf{p}'' \\ H_{\mathsf{p}'} + H_{\mathsf{p}''} - H_{\mathsf{p}'} \times H_{\mathsf{p}''} & \text{if } \mathsf{p} = \mathsf{p}' \vee \mathsf{p}'' \end{cases} \tag{B.10}$$

is characteristic function and $\mathsf{p}, \mathsf{p}', \mathsf{p}''$ are propositional formulas.

In order to find the energy function of the XOR $x \oplus y \leftrightarrow z$ we convert it to a formula that contains only conjunctives and disjunctives, as belows.

$$\begin{aligned} x \oplus y \leftrightarrow z &= (x \wedge \neg y) \vee (\neg x \wedge y) \leftrightarrow z \\ &= (((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge z) \vee (\neg((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge \neg z) \\ &= (((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge z) \vee (\neg(x \wedge \neg y) \wedge \neg(\neg x \wedge y) \wedge \neg z) \end{aligned} \tag{B.11}$$

The characteristic function:

$$H_{x\oplus y\leftrightarrow z} = H_{(((x\wedge\neg y)\vee(\neg x\wedge y))\wedge z)} + H_{(\neg(x\wedge\neg y)\wedge\neg(\neg x\wedge y)\wedge\neg z)} - H_{(((x\wedge\neg y)\vee(\neg x\wedge y))\wedge z)}H_{(\neg(x\wedge\neg y)\wedge\neg(\neg x\wedge y)\wedge\neg z)} \tag{B.12}$$

with[1]

$$\begin{aligned} H_{(((x\wedge\neg y)\vee(\neg x\wedge y))\wedge z)} &= (x(1-y) + y(1-x) - xy(1-x)(1-y))z \\ &= (x + y - 2xy)z \end{aligned} \tag{B.13}$$

[1]Note that $xy \times x = xy$

and

$$\begin{aligned} H_{(\neg(x\wedge\neg y)\wedge\neg(\neg x\wedge y)\wedge\neg z)} &= (1-x(1-y))(1-y(1-x))(1-z) \\ &= (1-(x+y-2xy))(1-z) \end{aligned} \tag{B.14}$$

apply Eq. B.13 and Eq. B.14 to Eq. B.12 we have:

$$H_{x\oplus y\leftrightarrow z} = 1-4xyz+2xy+2xz+2yz-x-y-z \tag{B.15}$$

and the energy function

$$E(x,y,z) = 4xyz-2xy-2xz-2yz+x+y+z \tag{B.16}$$

The high-order term xyz in this function can be replaced by:

$$xyz = min_h(xy-2xh-2yh+2zh+3h) \tag{B.17}$$

with h is an additional energy function and min_h is a function of h that returns a minimum value. The energy function in quadratic term then becomes: $E(x,y,z) = min_h(2xy-2xz-2yz-8xh-8yh+8zh+x+y+z+12h)$.

Appendix C

Rules from Car Valuation

$$\begin{aligned}
&h_1 \leftrightarrow safety\_is\_low \wedge the\_car\_is\_unacceptable \\
&h_2 \leftrightarrow can\_carry\_\_2\_people \wedge the\_car\_is\_unacceptable \\
&h_3 \leftrightarrow buying\_price\_is\_high \wedge maintenance\_price\_is\_very\_high \wedge the\_car\_is\_unacceptable \\
&h_4 \leftrightarrow buying\_price\_is\_very\_high \wedge maintenance\_price\_is\_high \wedge the\_car\_is\_unacceptable \\
&h_5 \leftrightarrow buying\_price\_is\_very\_high \wedge maintenance\_price\_is\_very\_high \wedge the\_car\_is\_unacceptable \\
&h_6 \leftrightarrow no\_of\_doors\_is\_2 \wedge can\_carry\_\_more\_than\_4\_people \wedge luggage\_boot\_size\_is\_small \wedge the\_car\_is\_unacceptable \\
&h_7 \leftrightarrow buying\_price\_is\_high \wedge luggage\_boot\_size\_is\_small \wedge safety\_is\_medium \wedge the\_car\_is\_unacceptable \\
&h_8 \leftrightarrow buying\_price\_is\_high \wedge maintenance\_price\_is\_high \wedge can\_carry\_\_4\_people \wedge safety\_is\_high \wedge the\_car\_is\_acceptable \\
&h_9 \leftrightarrow buying\_price\_is\_very\_high \wedge maintenance\_price\_is\_low \wedge luggage\_boot\_size\_is\_small \wedge safety\_is\_medium \wedge the\_car\_is\_unacceptable
\end{aligned} \tag{C.1}$$

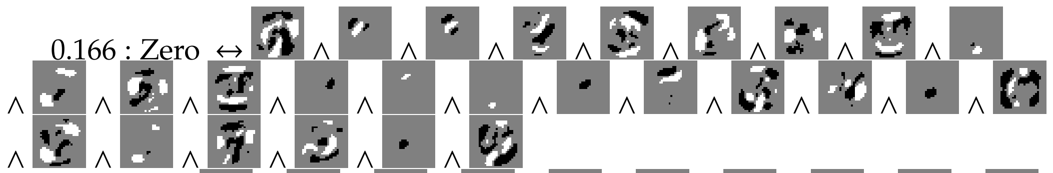
$$
\begin{aligned}
&h_{10} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_high} \wedge \mathit{can_carry__4_people} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_acceptable}\\
&h_{11} \leftrightarrow \mathit{maintenance_price_is_very_high} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_unacceptable}\\
&h_{12} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_high} \wedge \mathit{can_carry__4_people} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_acceptable}\\
&h_{13} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_very_high} \wedge \mathit{can_carry__4_people} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_acceptable}\\
&h_{14} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_high} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_unacceptable}\\
&h_{15} \leftrightarrow \mathit{buying_price_is_very_high} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_unacceptable}\\
&h_{16} \leftrightarrow \mathit{buying_price_is_very_high} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_acceptable}\\
&h_{17} \leftrightarrow \mathit{maintenance_price_is_very_high} \wedge \mathit{no_of_doors_is_2} \wedge \mathit{luggage_boot_size_is_medium} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_unacceptable}\\
&h_{18} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{19} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_good}\\
&h_{20} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{21} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_good}\\
&h_{22} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{23} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_high} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{24} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_good}\\
&h_{25} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{26} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_good}\\
&h_{27} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_good}\\
&h_{28} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_good}\\
&h_{29} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_acceptable}\\
&h_{30} \leftrightarrow \mathit{maintenance_price_is_very_high} \wedge \mathit{no_of_doors_is_2} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_unacceptable}\\
&h_{31} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_low} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_medium} \wedge \mathit{the_car_is_acceptable}\\
&h_{32} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_small} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_good}\\
&h_{33} \leftrightarrow \mathit{buying_price_is_low} \wedge \mathit{maintenance_price_is_very_high} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_medium} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_acceptable}\\
&h_{34} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__more_than_4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}\\
&h_{35} \leftrightarrow \mathit{buying_price_is_medium} \wedge \mathit{maintenance_price_is_medium} \wedge \mathit{can_carry__4_people} \wedge \mathit{luggage_boot_size_is_big} \wedge \mathit{safety_is_high} \wedge \mathit{the_car_is_very_good}
\end{aligned} \tag{C.2}
$$

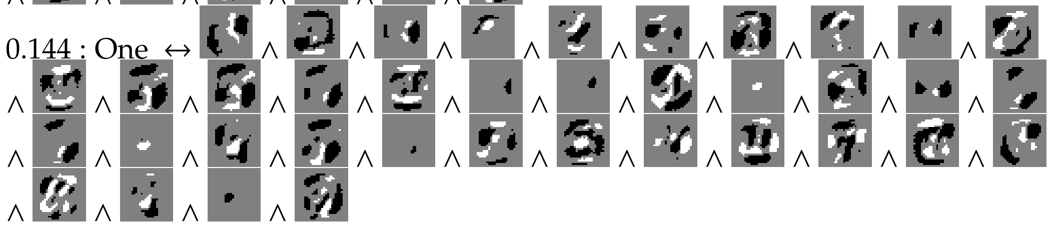
$$\begin{aligned}
&h_{36} \leftrightarrow buying_price_is_medium \wedge maintenance_price_is_medium \wedge no_of_doors_is_4 \wedge can_carry__more_than_4_people \wedge safety_is_medium \wedge the_car_is_acceptable\\
&h_{37} \leftrightarrow buying_price_is_medium \wedge maintenance_price_is_very_high \wedge can_carry__more_than_4_people \wedge luggage_boot_size_is_big \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{38} \leftrightarrow buying_price_is_medium \wedge maintenance_price_is_very_high \wedge can_carry__4_people \wedge luggage_boot_size_is_small \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{39} \leftrightarrow buying_price_is_medium \wedge maintenance_price_is_very_high \wedge no_of_doors_is_2 \wedge can_carry__4_people \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{40} \leftrightarrow buying_price_is_high \wedge maintenance_price_is_high \wedge can_carry__more_than_4_people \wedge luggage_boot_size_is_big \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{41} \leftrightarrow buying_price_is_high \wedge maintenance_price_is_high \wedge can_carry__4_people \wedge luggage_boot_size_is_big \wedge safety_is_medium \wedge the_car_is_acceptable\\
&h_{42} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_low \wedge can_carry__more_than_4_people \wedge luggage_boot_size_is_big \wedge safety_is_medium \wedge the_car_is_acceptable\\
&h_{43} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_low \wedge no_of_doors_is_4 \wedge can_carry__more_than_4_people \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{44} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_3 \wedge can_carry__more_than_4_people \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{45} \leftrightarrow buying_price_is_low \wedge maintenance_price_is_very_high \wedge no_of_doors_is_4 \wedge luggage_boot_size_is_small \wedge safety_is_medium \wedge the_car_is_unacceptable \qquad (C.3)\\
&h_{46} \leftrightarrow buying_price_is_very_high \wedge no_of_doors_is_4 \wedge can_carry__more_than_4_people \wedge luggage_boot_size_is_small \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{47} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_3 \wedge luggage_boot_size_is_small \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{48} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_2 \wedge luggage_boot_size_is_medium \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{49} \leftrightarrow buying_price_is_medium \wedge maintenance_price_is_very_high \wedge no_of_doors_is_more_than_5 \wedge luggage_boot_size_is_small \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{50} \leftrightarrow buying_price_is_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_more_than_5 \wedge can_carry__4_people \wedge safety_is_high \wedge the_car_is_acceptable\\
&h_{51} \leftrightarrow buying_price_is_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_2 \wedge luggage_boot_size_is_medium \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{52} \leftrightarrow buying_price_is_very_high \wedge no_of_doors_is_3 \wedge can_carry__4_people \wedge luggage_boot_size_is_medium \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{53} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_low \wedge no_of_doors_is_2 \wedge luggage_boot_size_is_medium \wedge safety_is_medium \wedge the_car_is_unacceptable\\
&h_{54} \leftrightarrow buying_price_is_very_high \wedge maintenance_price_is_medium \wedge no_of_doors_is_4 \wedge luggage_boot_size_is_small \wedge safety_is_medium \wedge the_car_is_unacceptable
\end{aligned}$$

Appendix D

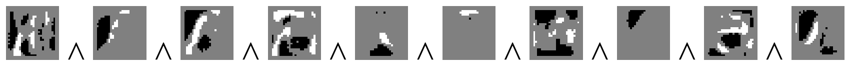
Visualisation of rules

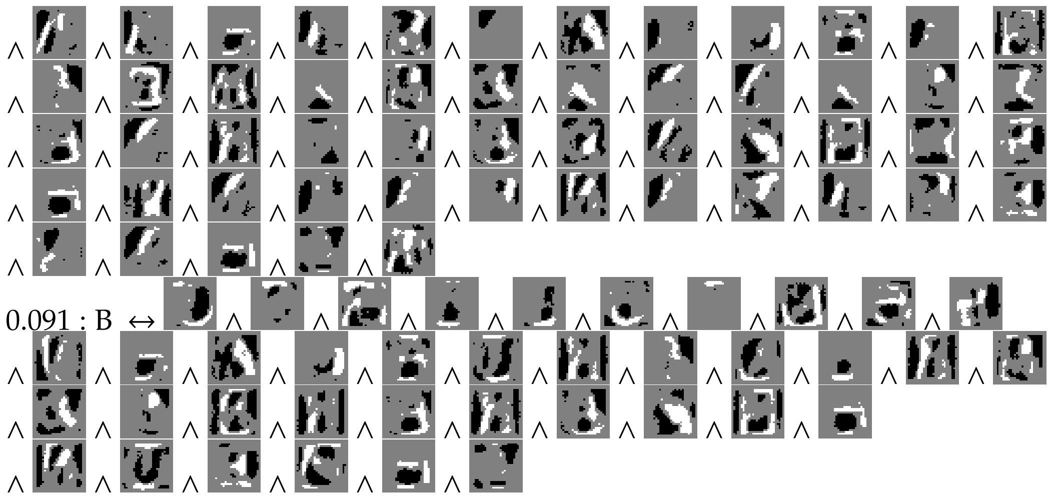
At first the rules from RBMs trained on 20000 MNIST Handwritten digit samples are extracted. Here, each image box represents the visualisation of rule from beliefs to a hypothesis. The white pixels represent positive literals, the dark pixels represent negative literals, the grey ones represent missing literals. We then extract rules from fist hidden layer to the label layer using TOP_RBM_EXTRACT. Since each literal in the top layer extraction represent the hypothesis in lower layer, therefore we replace the higher literals by the visualisation of lower rules. For the ease of presentation we only show the rules in lower layer whose hypotheses are positive literals in top layer.

0.166 : Zero ↔


0.144 : One ↔


The following shows the rules extracted from TiCC handwritten character dataset.

0.102 : A ↔



0.091 : B ↔

Appendix E

Visualisation of Representation Ranking

We generated the visualisation by normalising all basis vectors (feature detectors) $\mathbf{w}_j$ (column vector in the weight matrix) of an RBM to $[0 \quad 1]$ and reshaping each vector to a 2-dimensional image. In order to treat all basis vectors equally we used min,max method for normalisation with min, max were the minimum and maximum elements of the weight matrix. After representation ranking, we visualised the basis vectors in descending order of the scores. The order was from left to right and then from top to bottom for ease of presentation. In order to visualise large number of basis vectors, e.g. the whole hidden units of a network, it would be easier to see the ranking through the bases organised from top to bottom and then from left to right as in Figure E.1 and Figure E.2. Figure E.1 shows the basis vectors trained on 2000 face images from Frey faces dataset[1] (zoom in the Figure for better view). In Figure E.2 we show the basis vectors trained on handwritten letters data from TiCC collection.

[1]`http://www.cs.nyu.edu/~roweis/data.html`

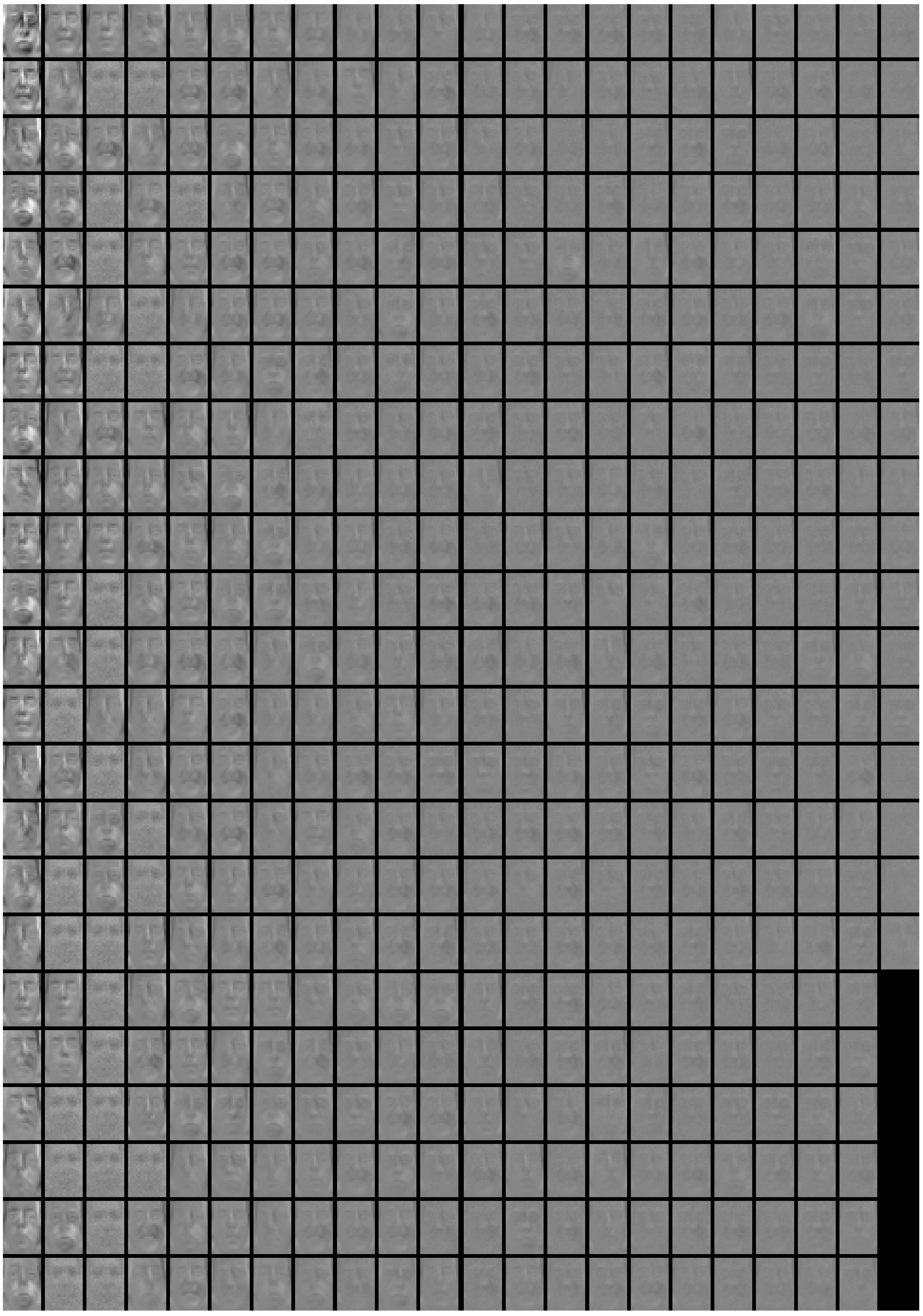


Figure E.1: Filter bases from RBM trained on Frey face images. The RBM has 500 units in hidden layer, the learning rate $\eta = 0.01$, sparsity gain $\lambda = 0.1$ ($p = 0.00001$). The bases are organised in descending order of their scores from top to bottom and from left to right.

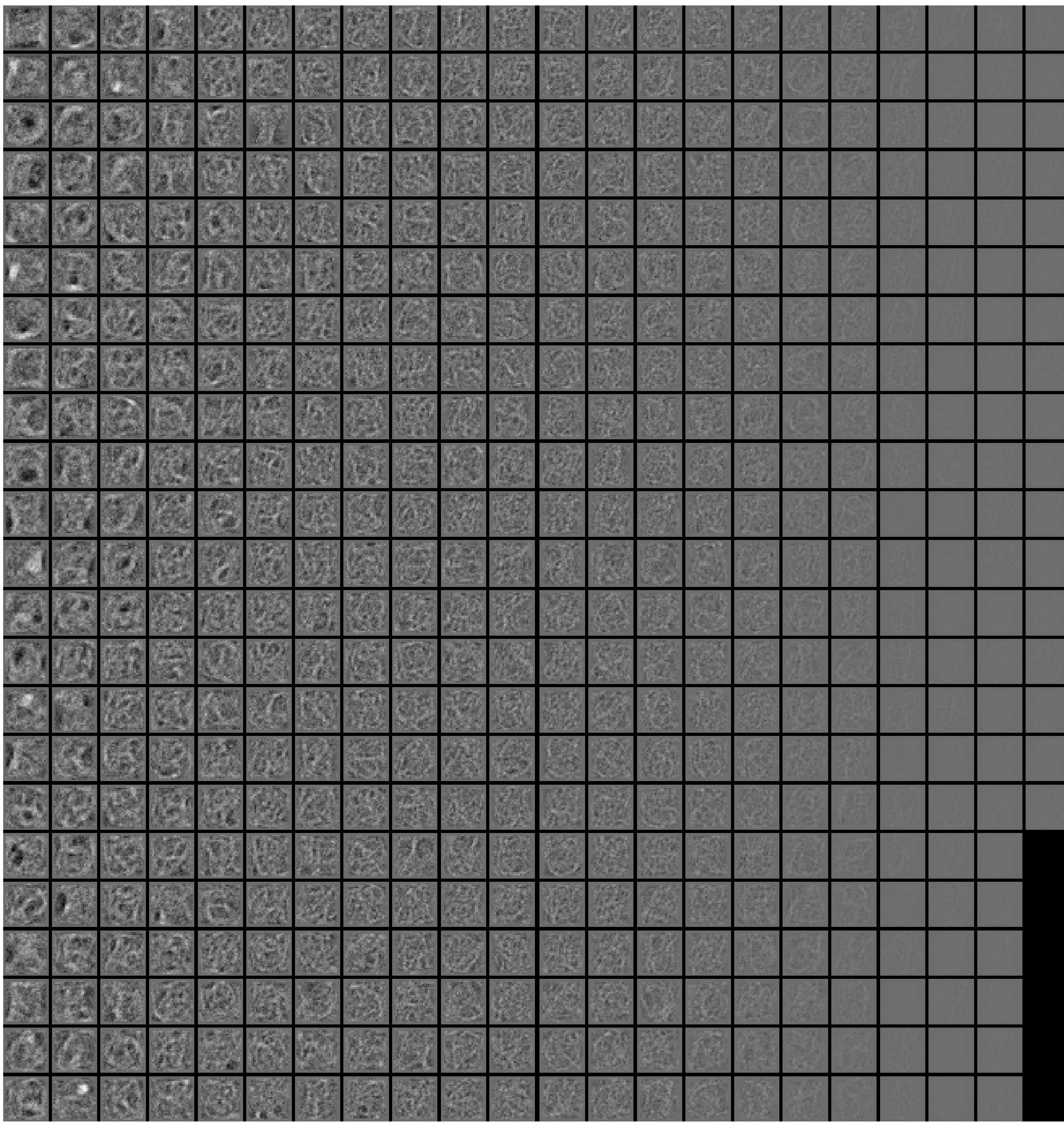


Figure E.2: Filter bases from RBM trained on handwritten letters (from A to Z). The RBM has 500 units in hidden layer, the learning rate $\eta = 0.3$, sparsity gain $\lambda = 0$ (no sparsity constraint). The bases are organised in descending order of their scores from top to bottom and from left to right.

Appendix F

Domain Adaptation for Sentiment Analysis

In this experiment we use the benchmark version of sentiment dataset of [14]. The data contains positive/negative reviews on Books, DVD, Electronics and Kitchen, and each has 2000 samples. The source data consist of 18,668 samples of unlabelled reviews, also from this dataset. The data is encoded as 2000-dimension vectors representing the presence/absence of the most frequent uni-grams and bi-grams in the vocabulary. We use RBMs with rectifier units [93] for representation knowledge transfer and a linear SVM as classifier. The reason behind this is the rectifier units can produce sparse representations that are well suited for linear classifier for sentiment analysis [46]. Experiment evaluation is done by using 10-fold cross validation.

Similar to previous experiments we apply the raw data of all four domains to learn the linear classifier which obtains similar results as in [37, 14]. Three transfer techniques are: RBM MIX - transfer source representation to combine with new representation learned in target domain; RBM STL (Transfer Data) - transfer the data from the source domain to combine with data in the target domain to learn the representation; and our adaptive transferred-profile that transfers the complete-models. With rectifier RBMs, approximation of mutual information becomes difficult due to the infinite number of binary units in the hidden layer. Also, the domain information captured in the rectifier RBMs seems to be distributed all over the feature detectors that would result in significant performance loss if pruning is used in the transfer. Therefore, we employ dropout to avoid biased sampling for the learning in the target domain.

	Books	DVD	Electronics	Kitchen
Linear SVM	80.65	80.85	85.05	85.95
RBM MIX	81.06	80.85	85.53	87.21
RBM STL (Transfer Data)	81.21	81.18	85.59	87.25
aTPL	81.55	81.89	86.17	87.29

Table F.1: Performance of transfer data (RBM STL) and transfer representation (RBM MIX & aTPL) on Books, DVD, Electronics, Kitchen domains of Amazon sentiment dataset.

The averaged results in Table F.1 may suggest that it is promising to share the knowledge which has already learned from a model. Even with the combination of representation knowledge RBM MIX achieved similar results as RBM STL in two domains Electronics and Kitchen. The aTPL achieve highest performance in all four domains.