Foundations and Trends® in
Machine Learning
Vol. 2, No. 1 (2009) 1–127

DOI: 10.1561/2200000006

now
the essence of knowledge

Learning Deep Architectures for AI

By Yoshua Bengio

Contents

Foundations and Trends® in
Machine Learning
Vol. 2, No. 1 (2009) 1–127

DOI: 10.1561/2200000006

now
the essence of knowledge

Learning Deep Architectures for AI

Yoshua Bengio

Dept. IRO, Université de Montréal, C.P. 6128, Montreal, Qc, H3C 3J7, Canada, yoshua.bengio@umontreal.ca

Abstract

Theoretical results suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g., in vision, language, and other AI-level tasks), one may need *deep architectures*. Deep architectures are composed of multiple levels of non-linear operations, such as in neural nets with many hidden layers or in complicated propositional formulae re-using many sub-formulae. Searching the parameter space of deep architectures is a difficult task, but learning algorithms such as those for Deep Belief Networks have recently been proposed to tackle this problem with notable success, beating the state-of-the-art in certain areas. This monograph discusses the motivations and principles regarding learning algorithms for deep architectures, in particular those exploiting as building blocks unsupervised learning of single-layer models such as Restricted Boltzmann Machines, used to construct deeper models such as Deep Belief Networks.

1

Introduction

Allowing computers to model our world well enough to exhibit what we call intelligence has been the focus of more than half a century of research. To achieve this, it is clear that a large quantity of information about our world should somehow be stored, explicitly or implicitly, in the computer. Because it seems daunting to formalize manually all that information in a form that computers can use to answer questions and generalize to new contexts, many researchers have turned to *learning algorithms* to capture a large fraction of that information. Much progress has been made to understand and improve learning algorithms, but the challenge of artificial intelligence (AI) remains. Do we have algorithms that can understand scenes and describe them in natural language? Not really, except in very limited settings. Do we have algorithms that can infer enough semantic concepts to be able to interact with most humans using these concepts? No. If we consider image understanding, one of the best specified of the AI tasks, we realize that we do not yet have learning algorithms that can discover the many visual and semantic concepts that would seem to be necessary to interpret most images on the web. The situation is similar for other AI tasks.

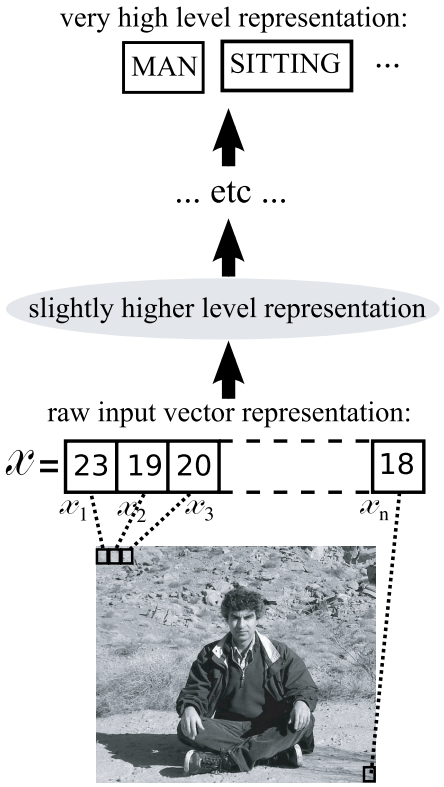

Fig. 1.1 We would like the raw input image to be transformed into gradually higher levels of representation, representing more and more abstract functions of the raw input, e.g., edges, local shapes, object parts, etc. In practice, we do not know in advance what the "right" representation should be for all these levels of abstractions, although linguistic concepts might help guessing what the higher levels should implicitly represent.

Consider for example the task of interpreting an input image such as the one in Figure 1.1. When humans try to solve a particular AI task (such as machine vision or natural language processing), they often exploit their intuition about how to decompose the problem into sub-problems and multiple levels of representation, e.g., in object parts and constellation models [138, 179, 197] where models for parts can be re-used in different object instances. For example, the current state-of-the-art in machine vision involves a sequence of modules starting from pixels and ending in a linear or kernel classifier [134, 145], with intermediate modules mixing engineered transformations and learning,

e.g., first extracting low-level features that are invariant to small geometric variations (such as edge detectors from Gabor filters), transforming them gradually (e.g., to make them invariant to contrast changes and contrast inversion, sometimes by pooling and sub-sampling), and then detecting the most frequent patterns. A plausible and common way to extract useful information from a natural image involves transforming the raw pixel representation into gradually more abstract representations, e.g., starting from the presence of edges, the detection of more complex but local shapes, up to the identification of abstract categories associated with sub-objects and objects which are parts of the image, and putting all these together to capture enough understanding of the scene to answer questions about it.

Here, we assume that the computational machinery necessary to express complex behaviors (which one might label "intelligent") requires *highly varying* mathematical functions, i.e., mathematical functions that are highly non-linear in terms of raw sensory inputs, and display a very large number of variations (ups and downs) across the domain of interest. We view the raw input to the learning system as a high dimensional entity, made of many observed variables, which are related by unknown intricate statistical relationships. For example, using knowledge of the 3D geometry of solid objects and lighting, we can relate small variations in underlying physical and geometric factors (such as position, orientation, lighting of an object) with changes in pixel intensities for all the pixels in an image. We call these *factors of variation* because they are different aspects of the data that can vary separately and often independently. In this case, explicit knowledge of the physical factors involved allows one to get a picture of the mathematical form of these dependencies, and of the shape of the set of images (as points in a high-dimensional space of pixel intensities) associated with the same 3D object. If a machine captured the factors that explain the statistical variations in the data, and how they interact to generate the kind of data we observe, we would be able to say that the machine *understands* those aspects of the world covered by these factors of variation. Unfortunately, in general and for most factors of variation underlying natural images, we do not have an analytical understanding of these factors of variation. We do not have enough formalized

prior knowledge about the world to explain the observed variety of images, even for such an apparently simple abstraction as **MAN**, illustrated in Figure 1.1. A high-level abstraction such as **MAN** has the property that it corresponds to a very large set of possible images, which might be very different from each other from the point of view of simple Euclidean distance in the space of pixel intensities. The set of images for which that label could be appropriate forms a highly convoluted region in pixel space that is not even necessarily a connected region. The **MAN** category can be seen as a high-level abstraction with respect to the space of images. What we call abstraction here can be a category (such as the **MAN** category) or a *feature*, a function of sensory data, which can be discrete (e.g., the input sentence is at the past tense) or continuous (e.g., the input video shows an object moving at 2 meter/second). Many lower-level and intermediate-level concepts (which we also call abstractions here) would be useful to construct a **MAN**-detector. Lower level abstractions are more directly tied to particular percepts, whereas higher level ones are what we call "more abstract" because their connection to actual percepts is more remote, and through other, intermediate-level abstractions.

In addition to the difficulty of coming up with the appropriate intermediate abstractions, the number of visual and semantic categories (such as **MAN**) that we would like an "intelligent" machine to capture is rather large. The focus of deep architecture learning is to automatically discover such abstractions, from the lowest level features to the highest level concepts. Ideally, we would like learning algorithms that enable this discovery with as little human effort as possible, i.e., without having to manually define all necessary abstractions or having to provide a huge set of relevant hand-labeled examples. If these algorithms could tap into the huge resource of text and images on the web, it would certainly help to transfer much of human knowledge into machine-interpretable form.

1.1 How do We Train Deep Architectures?

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of

lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. This is especially important for higher-level abstractions, which humans often do not know how to specify explicitly in terms of raw sensory input. The ability to automatically learn powerful features will become increasingly important as the amount of data and range of applications to machine learning methods continues to grow.

Depth of architecture refers to the number of levels of composition of non-linear operations in the function learned. Whereas most current learning algorithms correspond to *shallow architectures* (1, 2 or 3 levels), the mammal brain is organized in a *deep architecture* [173] with a given input percept represented at multiple levels of abstraction, each level corresponding to a different area of cortex. Humans often describe such concepts in hierarchical ways, with multiple levels of abstraction. The brain also appears to process information through multiple stages of transformation and representation. This is particularly clear in the primate visual system [173], with its sequence of processing stages: detection of edges, primitive shapes, and moving up to gradually more complex visual shapes.

Inspired by the architectural depth of the brain, neural network researchers had wanted for decades to train deep multi-layer neural networks [19, 191], but no successful attempts were reported before 2006[1]: researchers reported positive experimental results with typically two or three levels (i.e., one or two hidden layers), but training deeper networks consistently yielded poorer results. Something that can be considered a *breakthrough* happened in 2006: Hinton et al. at University of Toronto introduced Deep Belief Networks (DBNs) [73], with a learning algorithm that greedily trains one layer at a time, exploiting an unsupervised learning algorithm for each layer, a Restricted Boltzmann Machine (RBM) [51]. Shortly after, related algorithms based on auto-encoders were proposed [17, 153], apparently exploiting the

[1] Except for neural networks with a special structure called convolutional networks, discussed in Section 4.5.

same principle: *guiding the training of intermediate levels of representation using unsupervised learning, which can be performed locally at each level.* Other algorithms for deep architectures were proposed more recently that exploit neither RBMs nor auto-encoders and that exploit the same principle [131, 202] (see Section 4).

Since 2006, deep networks have been applied with success not only in classification tasks [2, 17, 99, 111, 150, 153, 195], but also in regression [160], dimensionality reduction [74, 158], modeling textures [141], modeling motion [182, 183], object segmentation [114], information retrieval [154, 159, 190], robotics [60], natural language processing [37, 130, 202], and collaborative filtering [162]. Although auto-encoders, RBMs and DBNs can be trained with unlabeled data, in many of the above applications, they have been successfully used to initialize deep *supervised* feedforward neural networks applied to a specific task.

1.2 Intermediate Representations: Sharing Features and Abstractions Across Tasks

Since a deep architecture can be seen as the composition of a series of processing stages, the immediate question that deep architectures raise is: what kind of representation of the data should be found as the output of each stage (i.e., the input of another)? What kind of interface should there be between these stages? A hallmark of recent research on deep architectures is the focus on these intermediate representations: the success of deep architectures belongs to the representations learned in an unsupervised way by RBMs [73], ordinary auto-encoders [17], sparse auto-encoders [150, 153], or denoising auto-encoders [195]. These algorithms (described in more detail in Section 7.2) can be seen as learning to transform one representation (the output of the previous stage) into another, at each step maybe disentangling better the factors of variations underlying the data. As we discuss at length in Section 4, it has been observed again and again that once a good representation has been found at each level, it can be used to initialize and successfully train a deep neural network by supervised gradient-based optimization.

Each level of abstraction found in the brain consists of the "activation" (neural excitation) of a small subset of a large number of features that are, in general, not mutually exclusive. Because these features are not mutually exclusive, they form what is called a *distributed representation* [68, 156]: the information is not localized in a particular neuron but distributed across many. In addition to being distributed, it appears that the brain uses a representation that is *sparse*: only a around 1-4% of the neurons are active together at a given time [5, 113]. Section 3.2 introduces the notion of sparse distributed representation and Section 7.1 describes in more detail the machine learning approaches, some inspired by the observations of the sparse representations in the brain, that have been used to build deep architectures with sparse representations.

这段讲人脑的机制：distributed, sparse

Whereas dense distributed representations are one extreme of a spectrum, and sparse representations are in the middle of that spectrum, purely local representations are the other extreme. Locality of representation is intimately connected with the notion of *local generalization*. Many existing machine learning methods are *local in input space*: to obtain a learned function that behaves differently in different regions of data-space, they require different tunable parameters for each of these regions (see more in Section 3.1). Even though statistical efficiency is not necessarily poor when the number of tunable parameters is large, good generalization can be obtained only when adding some form of prior (e.g., that smaller values of the parameters are preferred). When that prior is not task-specific, it is often one that forces the solution to be very smooth, as discussed at the end of Section 3.1. In contrast to learning methods based on local generalization, the total number of patterns that can be distinguished using a distributed representation scales possibly exponentially with the dimension of the representation (i.e., the number of learned features). 这句没看懂

In many machine vision systems, learning algorithms have been limited to specific parts of such a processing chain. The rest of the design remains labor-intensive, which might limit the scale of such systems. On the other hand, a hallmark of what we would consider intelligent machines includes a large enough repertoire of concepts. Recognizing **MAN** is not enough. We need algorithms that can tackle a very large

set of such tasks and concepts. It seems daunting to manually define that many tasks, and learning becomes essential in this context. Furthermore, it would seem foolish not to exploit the underlying commonalities between these tasks and between the concepts they require. This has been the focus of research on *multi-task learning* [7, 8, 32, 88, 186]. Architectures with multiple levels naturally provide such sharing and re-use of components: the low-level visual features (like edge detectors) and intermediate-level visual features (like object parts) that are useful to detect **MAN** are also useful for a large group of other visual tasks. Deep learning algorithms are based on learning intermediate representations which can be shared across tasks. Hence they can leverage unsupervised data and data from similar tasks [148] to boost performance on large and challenging problems that routinely suffer from a poverty of labelled data, as has been shown by [37], beating the state-of-the-art in several natural language processing tasks. A similar multi-task approach for deep architectures was applied in vision tasks by [2]. Consider a multi-task setting in which there are different outputs for different tasks, all obtained from a shared pool of high-level features. The fact that many of these learned features are shared among m tasks provides sharing of statistical strength in proportion to m. Now consider that these learned high-level features can themselves be represented by combining lower-level intermediate features from a common pool. Again statistical strength can be gained in a similar way, and this strategy can be exploited for every level of a deep architecture.

In addition, learning about a large set of interrelated concepts might provide a key to the kind of broad generalizations that humans appear able to do, which we would not expect from separately trained object detectors, with one detector per visual category. If each high-level category is itself represented through a particular distributed configuration of abstract features from a common pool, generalization to unseen categories could follow naturally from new configurations of these features. Even though only some configurations of these features would present in the training examples, if they represent different aspects of the data, new examples could meaningfully be represented by new configurations of these features.

1.3 Desiderata for Learning AI

Summarizing some of the above issues, and trying to put them in the broader perspective of AI, we put forward a number of requirements we believe to be important for learning algorithms to approach AI, many of which motivate the research are described here:

- Ability to learn complex, highly-varying functions, i.e., with a number of variations much greater than the number of training examples.
- Ability to learn with little human input the low-level, intermediate, and high-level abstractions that would be useful to represent the kind of complex functions needed for AI tasks.
- Ability to learn from a very large set of examples: computation time for training should scale well with the number of examples, i.e., close to linearly.
- Ability to learn from mostly unlabeled data, i.e., to work in the semi-supervised setting, where not all the examples come with complete and correct semantic labels.
- Ability to exploit the synergies present across a large number of tasks, i.e., multi-task learning. These synergies exist because all the AI tasks provide different views on the same underlying reality.
- Strong *unsupervised learning* (i.e., capturing most of the statistical structure in the observed data), which seems essential in the limit of a large number of tasks and when future tasks are not known ahead of time.

Other elements are equally important but are not directly connected to the material in this monograph. They include the ability to learn to represent context of varying length and structure [146], so as to allow machines to operate in a context-dependent stream of observations and produce a stream of actions, the ability to make decisions when actions influence the future observations and future rewards [181], and the ability to influence future observations so as to collect more relevant information about the world, i.e., a form of active learning [34].

1.4 Outline of the Paper

Section 2 reviews theoretical results (which can be skipped without hurting the understanding of the remainder) showing that an architecture with insufficient depth can require many more computational elements, potentially exponentially more (with respect to input size), than architectures whose depth is matched to the task. We claim that insufficient depth can be detrimental for learning. Indeed, if a solution to the task is represented with a very large but shallow architecture (with many computational elements), a lot of training examples might be needed to tune each of these elements and capture a highly varying function. Section 3.1 is also meant to motivate the reader, this time to highlight the limitations of local generalization and local estimation, which we expect to avoid using deep architectures with a distributed representation (Section 3.2).

In later sections, the monograph describes and analyzes some of the algorithms that have been proposed to train deep architectures. Section 4 introduces concepts from the neural networks literature relevant to the task of training deep architectures. We first consider the previous difficulties in training neural networks with many layers, and then introduce unsupervised learning algorithms that could be exploited to initialize deep neural networks. Many of these algorithms (including those for the RBM) are related to the *auto-encoder*: a simple unsupervised algorithm for learning a one-layer model that computes a distributed representation for its input [25, 79, 156]. To fully understand RBMs and many related unsupervised learning algorithms, Section 5 introduces the class of energy-based models, including those used to build generative models with hidden variables such as the Boltzmann Machine. Section 6 focuses on the greedy layer-wise training algorithms for Deep Belief Networks (DBNs) [73] and Stacked Auto-Encoders [17, 153, 195]. Section 7 discusses variants of RBMs and auto-encoders that have been recently proposed to extend and improve them, including the use of sparsity, and the modeling of temporal dependencies. Section 8 discusses algorithms for jointly training all the layers of a Deep Belief Network using variational bounds. Finally, we consider in Section 9 forward looking questions such as the hypothesized difficult optimization

problem involved in training deep architectures. In particular, we follow up on the hypothesis that part of the success of current learning strategies for deep architectures is connected to the optimization of lower layers. We discuss the principle of continuation methods, which minimize gradually less smooth versions of the desired cost function, to make a dent in the optimization of deep architectures.

2

Theoretical Advantages of Deep Architectures

In this section, we present a motivating argument for the study of learning algorithms for deep architectures, by way of theoretical results revealing potential limitations of architectures with insufficient depth. This part of the monograph (this section and the next) motivates the algorithms described in the later sections, and can be skipped without making the remainder difficult to follow.

The main point of this section is that some functions cannot be efficiently represented (in terms of number of tunable elements) by architectures that are too shallow. These results suggest that it would be worthwhile to explore learning algorithms for deep architectures, which might be able to represent some functions otherwise not efficiently representable. Where simpler and shallower architectures fail to efficiently represent (and hence to learn) a task of interest, we can hope for learning algorithms that could set the parameters of a deep architecture for this task.

We say that the expression of a function is *compact* when it has few computational elements, i.e., few degrees of freedom that need to be tuned by learning. So for a fixed number of training examples, and short of other sources of knowledge injected in the learning algorithm,

we would expect that compact representations of the target function[1] would yield better generalization.

More precisely, functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth $k-1$ architecture. Since the number of computational elements one can afford depends on the number of training examples available to tune or select them, the consequences are not only computational but also statistical: poor generalization may be expected when using an insufficiently deep architecture for representing some functions.

We consider the case of fixed-dimension inputs, where the computation performed by the machine can be represented by a directed acyclic graph where each node performs a computation that is the application of a function on its inputs, each of which is the output of another node in the graph or one of the external inputs to the graph. The whole graph can be viewed as a *circuit* that computes a function applied to the external inputs. When the set of functions allowed for the computation nodes is limited to *logic gates*, such as {AND, OR, NOT}, this is a Boolean circuit, or *logic circuit*.

To formalize the notion of depth of architecture, one must introduce the notion of a *set of computational elements*. An example of such a set is the set of computations that can be performed logic gates. Another is the set of computations that can be performed by an artificial neuron (depending on the values of its synaptic weights). A function can be expressed by the composition of computational elements from a given set. It is defined by a graph which formalizes this composition, with one node per computational element. Depth of architecture refers to the depth of that graph, i.e., the longest path from an input node to an output node. When the set of computational elements is the set of computations an artificial neuron can perform, depth corresponds to the number of layers in a neural network. Let us explore the notion of depth with examples of architectures of different depths. Consider the function $f(x) = x * \sin(a * x + b)$. It can be expressed as the composition of simple operations such as addition, subtraction, multiplication,

[1] The target function is the function that we would like the learner to discover.

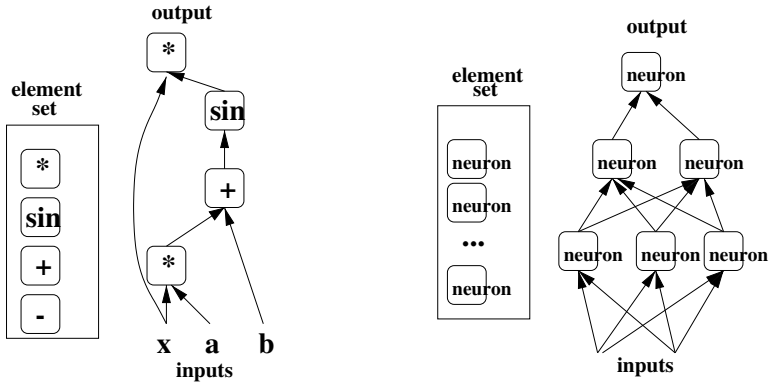

Fig. 2.1 Examples of functions represented by a graph of computations, where each node is taken in some "element set" of allowed computations. Left, the elements are $\{*, +, -, \sin\} \cup \mathbf{R}$. The architecture computes $x * \sin(a * x + b)$ and has depth 4. Right, the elements are artificial neurons computing $f(\mathbf{x}) = \tanh(b + \mathbf{w}'\mathbf{x})$; each element in the set has a different $(\mathbf{w}, b)$ parameter. The architecture is a multi-layer neural network of depth 3.

and the sin operation, as illustrated in Figure 2.1. In the example, there would be a different node for the multiplication $a * x$ and for the final multiplication by x. Each node in the graph is associated with an output value obtained by applying some function on input values that are the outputs of other nodes of the graph. For example, in a logic circuit each node can compute a Boolean function taken from a small set of Boolean functions. The graph as a whole has input nodes and output nodes and computes a function from input to output. The *depth* of an architecture is the maximum length of a path from any input of the graph to any output of the graph, i.e., 4 in the case of $x * \sin(a * x + b)$ in Figure 2.1.

- If we include affine operations and their possible composition with sigmoids in the set of computational elements, linear regression and logistic regression have depth 1, i.e., have a single level.
- When we put a fixed kernel computation $K(\mathbf{u}, \mathbf{v})$ in the set of allowed operations, along with affine operations, kernel machines [166] with a fixed kernel can be considered to have two levels. The first level has one element computing

$K(\mathbf{x}, \mathbf{x}_i)$ for each prototype $\mathbf{x}_i$ (a selected representative training example) and matches the input vector $\mathbf{x}$ with the prototypes $\mathbf{x}_i$. The second level performs an affine combination $b + \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i)$ to associate the matching prototypes $\mathbf{x}_i$ with the expected response.

- When we put artificial neurons (affine transformation followed by a non-linearity) in our set of elements, we obtain ordinary multi-layer neural networks [156]. With the most common choice of one hidden layer, they also have depth two (the hidden layer and the output layer).
- Decision trees can also be seen as having two levels, as discussed in Section 3.1.
- Boosting [52] usually adds one level to its base learners: that level computes a vote or linear combination of the outputs of the base learners.
- Stacking [205] is another meta-learning algorithm that adds one level.
- Based on current knowledge of brain anatomy [173], it appears that the cortex can be seen as a deep architecture, with 5–10 levels just for the visual system.

Although depth depends on the choice of the set of allowed computations for each element, graphs associated with one set can often be converted to graphs associated with another by an graph transformation in a way that multiplies depth. Theoretical results suggest that it is not the absolute number of levels that matters, but the number of levels relative to how many are required to represent efficiently the target function (with some choice of set of computational elements).

2.1 Computational Complexity

The most formal arguments about the power of deep architectures come from investigations into computational complexity of circuits. The basic conclusion that these results suggest is that *when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one.*

A two-layer circuit of logic gates can represent any Boolean function [127]. Any Boolean function can be written as a sum of products (disjunctive normal form: AND gates on the first layer with optional negation of inputs, and OR gate on the second layer) or a product of sums (conjunctive normal form: OR gates on the first layer with optional negation of inputs, and AND gate on the second layer). To understand the limitations of shallow architectures, the first result to consider is that with depth-two logical circuits, most Boolean functions require an *exponential* (with respect to input size) number of logic gates [198] to be represented.

More interestingly, there are functions computable with a polynomial-size logic gates circuit of depth k that require exponential size when restricted to depth $k-1$ [62]. The proof of this theorem relies on earlier results [208] showing that *d-bit parity circuits of depth 2 have exponential size*. The d-bit *parity function* is defined as usual:

$$\text{parity}: (b_1, \ldots, b_d) \in \{0,1\}^d \mapsto \begin{cases} 1, & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ 0, & \text{otherwise.} \end{cases}$$

One might wonder whether these computational complexity results for Boolean circuits are relevant to machine learning. See [140] for an early survey of theoretical results in computational complexity relevant to learning algorithms. Interestingly, many of the results for Boolean circuits can be generalized to architectures whose computational elements are *linear threshold* units (also known as artificial neurons [125]), which compute

$$f(\mathbf{x}) = \mathbf{1}_{\mathbf{w}'\mathbf{x}+b\geq 0} \tag{2.1}$$

with parameters $\mathbf{w}$ and b. The *fan-in* of a circuit is the maximum number of inputs of a particular element. Circuits are often organized in layers, like multi-layer neural networks, where elements in a layer only take their input from elements in the previous layer(s), and the first layer is the neural network input. The *size* of a circuit is the number of its computational elements (excluding input elements, which do not perform any computation).

Of particular interest is the following theorem, which applies to *monotone weighted threshold circuits* (i.e., multi-layer neural networks with linear threshold units and positive weights) when trying to represent a function compactly representable with a depth k circuit:

Theorem 2.1. A monotone weighted threshold circuit of depth $k-1$ computing a function $f_k \in \mathcal{F}_{k,N}$ has size at least 2^{cN} for some constant $c > 0$ and $N > N_0$ [63].

The class of functions $\mathcal{F}_{k,N}$ is defined as follows. It contains functions with N^{2k-2} inputs, defined by a depth k circuit that is a tree. At the leaves of the tree there are unnegated input variables, and the function value is at the root. The ith level from the bottom consists of AND gates when i is even and OR gates when i is odd. The fan-in at the top and bottom level is N and at all other levels it is N^2.

The above results do not prove that other classes of functions (such as those we want to learn to perform AI tasks) require deep architectures, nor that these demonstrated limitations apply to other types of circuits. However, these theoretical results beg the question: are the depth 1, 2 and 3 architectures (typically found in most machine learning algorithms) too shallow to represent efficiently more complicated functions of the kind needed for AI tasks? Results such as the above theorem also suggest that *there might be no universally right depth*: each function (i.e., each task) might require a particular minimum depth (for a given set of computational elements). We should therefore strive to develop learning algorithms that use the data to determine the depth of the final architecture. Note also that recursive computation defines a computation graph whose depth increases linearly with the number of iterations.

2.2 Informal Arguments

Depth of architecture is connected to the notion of highly varying functions. We argue that, in general, deep architectures can compactly represent highly varying functions which would otherwise require a very large size to be represented with an inappropriate architecture. We say

that a function is *highly varying* when a piecewise approximation (e.g., piecewise-constant or piecewise-linear) of that function would require a large number of pieces. A deep architecture is a composition of many operations, and it could in any case be represented by a possibly very large depth-2 architecture. The composition of computational units in a small but deep circuit can actually be seen as an efficient "factorization" of a large but shallow circuit. Reorganizing the way in which computational units are composed can have a drastic effect on the efficiency of representation size. For example, imagine a depth $2k$ representation of polynomials where odd layers implement products and even layers implement sums. This architecture can be seen as a particularly efficient factorization, which when expanded into a depth 2 architecture such as a sum of products, might require a huge number of terms in the sum: consider a level 1 product (like $\mathbf{x_2x_3}$ in Figure 2.2) from the depth $2k$ architecture. It could occur many times as a factor in many terms of the depth 2 architecture. One can see in this example that deep architectures can be advantageous if some computations (e.g., at one level) can be shared (when considering the expanded depth 2 expression): in that case, the overall expression to be represented can be factored out, i.e., represented more compactly with a deep architecture.

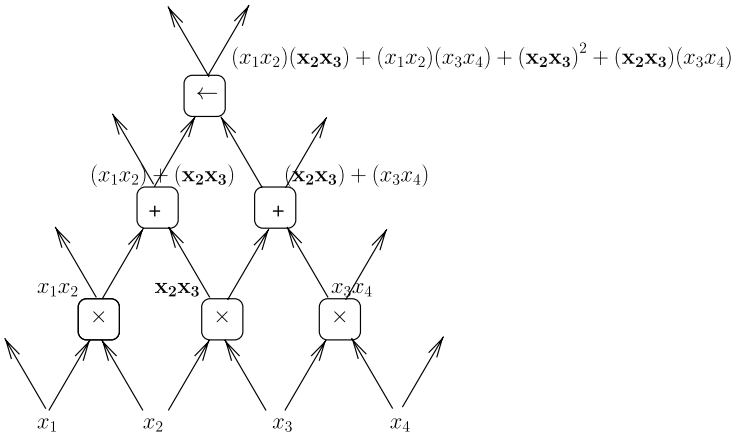

Fig. 2.2 Example of polynomial circuit (with products on odd layers and sums on even ones) illustrating the factorization enjoyed by a deep architecture. For example the level-1 product $\mathbf{x_2x_3}$ would occur many times (exponential in depth) in a depth 2 (sum of product) expansion of the above polynomial.

Further examples suggesting greater expressive power of deep architectures and their potential for AI and machine learning are also discussed by [19]. An earlier discussion of the expected advantages of deeper architectures in a more cognitive perspective is found in [191]. Note that connectionist cognitive psychologists have been studying for long time the idea of neural computation organized with a hierarchy of levels of representation corresponding to different levels of abstraction, with a distributed representation at each level [67, 68, 123, 122, 124, 157]. The modern deep architecture approaches discussed here owe a lot to these early developments. These concepts were introduced in cognitive psychology (and then in computer science / AI) in order to explain phenomena that were not as naturally captured by earlier cognitive models, and also to connect the cognitive explanation with the computational characteristics of the neural substrate.

To conclude, a number of computational complexity results strongly suggest that functions that can be compactly represented with a depth k architecture could require a very large number of elements in order to be represented by a shallower architecture. Since each element of the architecture might have to be selected, i.e., learned, using examples, these results suggest that depth of architecture can be very important from the point of view of statistical efficiency. This notion is developed further in the next section, discussing a related weakness of many shallow architectures associated with non-parametric learning algorithms: locality in input space of the estimator.

3

Local vs Non-Local Generalization

3.1 The Limits of Matching Local Templates

How can a learning algorithm compactly represent a "complicated" function of the input, i.e., one that has many more variations than the number of available training examples? This question is both connected to the depth question and to the question of locality of estimators. We argue that local estimators are inappropriate to learn highly varying functions, even though they can potentially be represented efficiently with deep architectures. An estimator that is *local in input space* obtains good generalization for a new input $\mathbf{x}$ by mostly exploiting training examples in the neighborhood of $\mathbf{x}$. For example, the k nearest neighbors of the test point $\mathbf{x}$, among the training examples, vote for the prediction at $\mathbf{x}$. Local estimators implicitly or explicitly partition the input space in regions (possibly in a soft rather than hard way) and require different parameters or degrees of freedom to account for the possible shape of the target function in each of the regions. When many regions are necessary because the function is highly varying, the number of required parameters will also be large, and thus the number of examples needed to achieve good generalization.

The local generalization issue is directly connected to the literature on the *curse of dimensionality*, but the results we cite show that *what matters for generalization is not dimensionality, but instead the number of "variations" of the function we wish to obtain after learning*. For example, if the function represented by the model is piecewise-constant (e.g., decision trees), then the question that matters is the number of pieces required to approximate properly the target function. There are connections between the number of variations and the input dimension: one can readily design families of target functions for which the number of variations is exponential in the input dimension, such as the parity function with d inputs.

Architectures based on matching local templates can be thought of as having two levels. The first level is made of a set of templates which can be matched to the input. A template unit will output a value that indicates the degree of matching. The second level combines these values, typically with a simple linear combination (an OR-like operation), in order to estimate the desired output. One can think of this linear combination as performing a kind of interpolation in order to produce an answer in the region of input space that is between the templates.

The prototypical example of architectures based on matching local templates is the *kernel machine* [166]

$$f(\mathbf{x}) = b + \sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i), \tag{3.1}$$

where b and α_i form the second level, while on the first level, the *kernel function* $K(\mathbf{x}, \mathbf{x}_i)$ matches the input $\mathbf{x}$ to the training example $\mathbf{x}_i$ (the sum runs over some or all of the input patterns in the training set). In the above equation, $f(\mathbf{x})$ could be for example, the discriminant function of a classifier, or the output of a regression predictor.

A kernel is *local* when $K(\mathbf{x}, \mathbf{x}_i) > \rho$ is true only for $\mathbf{x}$ in some connected region around $\mathbf{x}_i$ (for some threshold ρ). The size of that region can usually be controlled by a hyper-parameter of the kernel function. An example of local kernel is the Gaussian kernel $K(\mathbf{x}, \mathbf{x}_i) = e^{-||\mathbf{x}-\mathbf{x}_i||^2/\sigma^2}$, where σ controls the size of the region around $\mathbf{x}_i$. We can see the Gaussian kernel as computing a soft conjunction, because

it can be written as a product of one-dimensional conditions: $K(\mathbf{u},\mathbf{v}) = \prod_j e^{-(\mathbf{u}_j-\mathbf{v}_j)^2/\sigma^2}$. If $|\mathbf{u}_j - \mathbf{v}_j|/\sigma$ is small for all dimensions j, then the pattern matches and $K(\mathbf{u},\mathbf{v})$ is large. If $|\mathbf{u}_j - \mathbf{v}_j|/\sigma$ is large for a single j, then there is no match and $K(\mathbf{u},\mathbf{v})$ is small.

Well-known examples of kernel machines include not only Support Vector Machines (SVMs) [24, 39] and Gaussian processes [203] [1] for classification and regression, but also classical non-parametric learning algorithms for classification, regression and density estimation, such as the k-nearest neighbor algorithm, Nadaraya-Watson or Parzen windows density, regression estimators, etc. Below, we discuss *manifold learning algorithms* such as Isomap and LLE that can also be seen as local kernel machines, as well as related semi-supervised learning algorithms also based on the construction of a *neighborhood graph* (with one node per example and arcs between neighboring examples).

Kernel machines with a local kernel yield generalization by exploiting what could be called the *smoothness prior*: the assumption that the target function is smooth or can be well approximated with a smooth function. For example, in supervised learning, if we have the training example $(\mathbf{x}_i, y_i)$, then it makes sense to construct a predictor $f(\mathbf{x})$ which will output something close to y_i when $\mathbf{x}$ is close to $\mathbf{x}_i$. Note how this prior requires defining a notion of proximity in input space. This is a useful prior, but one of the claims made [13] and [19] is that such a prior is often insufficient to generalize when the target function is highly varying in input space.

The limitations of a fixed generic kernel such as the Gaussian kernel have motivated a lot of research in *designing kernels* based on prior knowledge about the task [38, 56, 89, 167]. However, if we lack sufficient prior knowledge for designing an appropriate kernel, can we learn it? This question also motivated much research [40, 96, 196], and deep architectures can be viewed as a promising development in this direction. It has been shown that a Gaussian Process kernel machine can be improved using a Deep Belief Network to learn a feature space [160]: after training the Deep Belief Network, its parameters are used to

[1] In the Gaussian Process case, as in kernel regression, $f(\mathbf{x})$ in Equation (3.1) is the conditional expectation of the target variable Y to predict, given the input $\mathbf{x}$.

initialize a deterministic non-linear transformation (a multi-layer neural network) that computes a feature vector (a new feature space for the data), and that transformation can be tuned to minimize the prediction error made by the Gaussian process, using a gradient-based optimization. The feature space can be seen as a learned representation of the data. Good representations bring close to each other examples which share abstract characteristics that are relevant factors of variation of the data distribution. Learning algorithms for deep architectures can be seen as ways to learn a good feature space for kernel machines.

Consider one direction $\mathbf{v}$ in which a target function f (what the learner should ideally capture) goes up and down (i.e., as α increases, $f(\mathbf{x} + \alpha\mathbf{v}) - b$ crosses 0, becomes positive, then negative, positive, then negative, etc.), in a series of "bumps". Following [165], [13, 19] show that for kernel machines with a Gaussian kernel, the required number of examples grows linearly with the number of bumps in the target function to be learned. They also show that for a maximally varying function such as the parity function, the number of examples necessary to achieve some error rate with a Gaussian kernel machine is *exponential in the input dimension*. For a learner that only relies on the prior that the target function is locally smooth (e.g., Gaussian kernel machines), learning a function with many sign changes in one direction is fundamentally difficult (requiring a large VC-dimension, and a correspondingly large number of examples). However, learning could work with other classes of functions in which the pattern of variations is captured compactly (a trivial example is when the variations are periodic and the class of functions includes periodic functions that approximately match).

For complex tasks in high dimension, the complexity of the decision surface could quickly make learning impractical when using a local kernel method. It could also be argued that if the curve has many variations and these variations are not related to each other through an underlying regularity, then no learning algorithm will do much better than estimators that are local in input space. However, it might be worth looking for more compact representations of these variations, because if one could be found, it would be likely to lead to better generalization, especially for variations not seen in the training set.

Of course this could only happen if there were underlying regularities to be captured in the target function; we expect this property to hold in AI tasks.

Estimators that are local in input space are found not only in supervised learning algorithms such as those discussed above, but also in unsupervised and semi-supervised learning algorithms, e.g., Locally Linear Embedding [155], Isomap [185], kernel Principal Component Analysis [168] (or kernel PCA) Laplacian Eigenmaps [10], Manifold Charting [26], spectral clustering algorithms [199], and kernel-based non-parametric semi-supervised algorithms [9, 44, 209, 210]. Most of these unsupervised and semi-supervised algorithms rely on the *neighborhood graph*: a graph with one node per example and arcs between near neighbors. With these algorithms, one can get a geometric intuition of what they are doing, as well as how being local estimators can hinder them. This is illustrated with the example in Figure 3.1 in the case of manifold learning. Here again, it was found that in order to

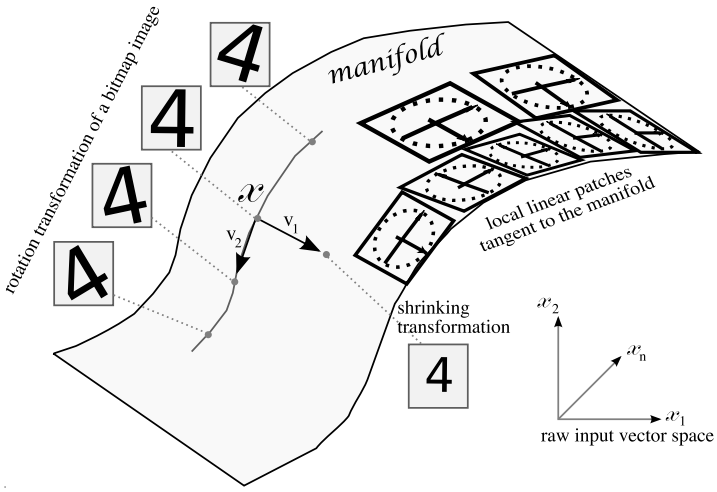

Fig. 3.1 The set of images associated with the same object class forms a manifold or a set of disjoint manifolds, i.e., regions of lower dimension than the original space of images. By rotating or shrinking, e.g., a digit 4, we get other images of the same class, i.e., on the same manifold. Since the manifold is locally smooth, it can in principle be approximated locally by linear patches, each being tangent to the manifold. Unfortunately, if the manifold is highly curved, the patches are required to be small, and exponentially many might be needed with respect to manifold dimension. Graph graciously provided by Pascal Vincent.

cover the many possible variations in the function to be learned, one needs a number of examples proportional to the number of variations to be covered [21].

Finally let us consider the case of semi-supervised learning algorithms based on the neighborhood graph [9, 44, 209, 210]. These algorithms partition the neighborhood graph in regions of constant label. It can be shown that the number of regions with constant label cannot be greater than the number of labeled examples [13]. Hence one needs at least as many labeled examples as there are variations of interest for the classification. This can be prohibitive if the decision surface of interest has a very large number of variations.

Decision trees [28] are among the best studied learning algorithms. Because they can focus on specific subsets of input variables, at first blush they seem non-local. However, they are also local estimators in the sense of relying on a partition of the input space and using separate parameters for each region [14], with each region associated with a leaf of the decision tree. This means that they also suffer from the limitation discussed above for other non-parametric learning algorithms: they need at least as many training examples as there are variations of interest in the target function, and they cannot generalize to new variations not covered in the training set. Theoretical analysis [14] shows specific classes of functions for which the number of training examples necessary to achieve a given error rate is exponential in the input dimension. This analysis is built along lines similar to ideas exploited previously in the computational complexity literature [41]. These results are also in line with previous empirical results [143, 194] showing that the generalization performance of decision trees degrades when the number of variations in the target function increases.

Ensembles of trees (like boosted trees [52], and forests [80, 27]) are more powerful than a single tree. They add a third level to the architecture which allows the model to discriminate among a number of regions *exponential in the number of parameters* [14]. As illustrated in Figure 3.2, they implicitly form a *distributed representation* (a notion discussed further in Section 3.2) with the output of all the trees in the forest. Each tree in an ensemble can be associated with a discrete symbol identifying the leaf/region in which the input example falls for

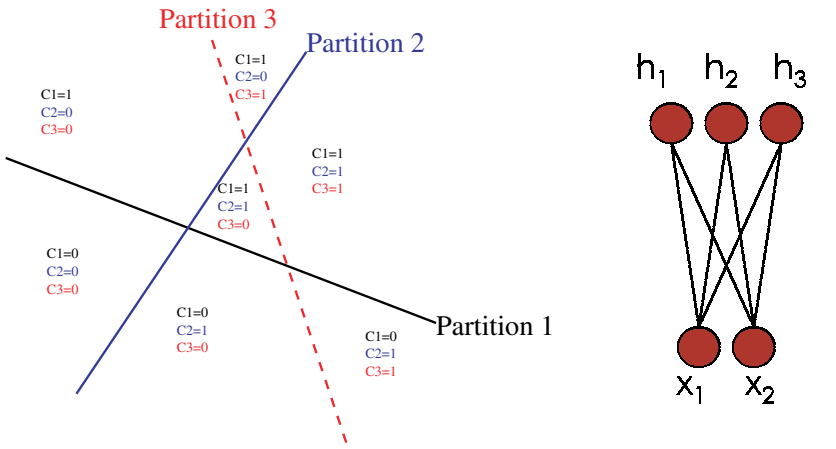

Fig. 3.2 Whereas a single decision tree (here just a two-way partition) can discriminate among a number of regions linear in the number of parameters (leaves), an ensemble of trees (*left*) can discriminate among a number of regions exponential in the number of trees, i.e., exponential in the total number of parameters (at least as long as the number of trees does not exceed the number of inputs, which is not quite the case here). Each distinguishable region is associated with one of the leaves of each tree (here there are three 2-way trees, each defining two regions, for a total of seven regions). This is equivalent to a multi-clustering, here three clusterings each associated with two regions. A binomial RBM with three hidden units (*right*) is a multi-clustering with 2 linearly separated regions per partition (each associated with one of the three binomial hidden units). A multi-clustering is therefore a distributed representation of the input pattern.

that tree. The identity of the leaf node in which the input pattern is associated for each tree forms a tuple that is a very rich description of the input pattern: it can represent a very large number of possible patterns, because the number of intersections of the leaf regions associated with the n trees can be exponential in n.

3.2 Learning Distributed Representations

In Section 1.2, we argued that deep architectures call for making choices about the kind of representation at the interface between levels of the system, and we introduced the basic notion of local representation (discussed further in the previous section), of distributed representation, and of sparse distributed representation. The idea of distributed representation is an old idea in machine learning and neural networks research [15, 68, 128, 157, 170], and it may be of help in dealing with

the curse of dimensionality and the limitations of local generalization. A cartoon *local representation* for integers $i \in \{1, 2, \ldots, N\}$ is a vector $\mathbf{r}(i)$ of N bits with a single 1 and $N-1$ zeros, i.e., with jth element $\mathbf{r}_j(i) = \mathbf{1}_{i=j}$, called the *one-hot* representation of i. A distributed representation for the same integer could be a vector of $\log_2 N$ bits, which is a much more compact way to represent i. For the same number of possible configurations, a distributed representation can potentially be exponentially more compact than a very local one. Introducing the notion of *sparsity* (e.g., encouraging many units to take the value 0) allows for representations that are in between being fully local (i.e., maximally sparse) and non-sparse (i.e., dense) distributed representations. Neurons in the cortex are believed to have a distributed and sparse representation [139], with around 1-4% of the neurons active at any one time [5, 113]. In practice, we often take advantage of representations which are continuous-valued, which increases their expressive power. An example of continuous-valued local representation is one where the ith element varies according to some distance between the input and a prototype or region center, as with the Gaussian kernel discussed in Section 3.1. In a distributed representation the input pattern is represented by a set of features that are not mutually exclusive, and might even be statistically independent. For example, clustering algorithms do not build a distributed representation since the clusters are essentially mutually exclusive, whereas Independent Component Analysis (ICA) [11, 142] and Principal Component Analysis (PCA) [82] build a distributed representation.

Consider a discrete distributed representation $\mathbf{r}(\mathbf{x})$ for an input pattern $\mathbf{x}$, where $\mathbf{r}_i(\mathbf{x}) \in \{1, \ldots M\}$, $i \in \{1, \ldots, N\}$. Each $\mathbf{r}_i(\mathbf{x})$ can be seen as a classification of $\mathbf{x}$ into M classes. As illustrated in Figure 3.2 (with $M = 2$), each $\mathbf{r}_i(\mathbf{x})$ partitions the $\mathbf{x}$-space in M regions, but the different partitions can be combined to give rise to a potentially exponential number of possible intersection regions in $\mathbf{x}$-space, corresponding to different configurations of $\mathbf{r}(\mathbf{x})$. Note that when representing a particular input distribution, some configurations may be impossible because they are incompatible. For example, in language modeling, a local representation of a word could directly encode its identity by an index in the vocabulary table, or equivalently a one-hot code with as many

entries as the vocabulary size. On the other hand, a distributed representation could represent the word by concatenating in one vector indicators for syntactic features (e.g., distribution over parts of speech it can have), morphological features (which suffix or prefix does it have?), and semantic features (is it the name of a kind of animal? etc). Like in clustering, we construct discrete classes, but the potential number of combined classes is huge: we obtain what we call a *multi-clustering* and that is similar to the idea of overlapping clusters and partial memberships [65, 66] in the sense that cluster memberships are not mutually exclusive. Whereas clustering forms a single partition and generally involves a heavy loss of information about the input, a multi-clustering provides a *set* of separate partitions of the input space. Identifying which region of each partition the input example belongs to forms a description of the input pattern which might be very rich, possibly not losing any information. The tuple of symbols specifying which region of each partition the input belongs to can be seen as a transformation of the input into a new space, where the statistical structure of the data and the factors of variation in it could be disentangled. This corresponds to the kind of partition of $\mathbf{x}$-space that an ensemble of trees can represent, as discussed in the previous section. This is also what we would like a deep architecture to capture, but with multiple levels of representation, the higher levels being more abstract and representing more complex regions of input space.

In the realm of supervised learning, multi-layer neural networks [157, 156] and in the realm of unsupervised learning, Boltzmann machines [1] have been introduced with the goal of learning distributed internal representations in the hidden layers. Unlike in the linguistic example above, the objective is to let learning algorithms discover the features that compose the distributed representation. In a multi-layer neural network with more than one hidden layer, there are several representations, one at each layer. Learning multiple levels of distributed representations involves a challenging training problem, which we discuss next.

4

Neural Networks for Deep Architectures

4.1 Multi-Layer Neural Networks

A typical set of equations for multi-layer neural networks [156] is the following. As illustrated in Figure 4.1, layer k computes an output vector $\mathbf{h}^k$ using the output $\mathbf{h}^{k-1}$ of the previous layer, starting with the input $\mathbf{x} = \mathbf{h}^0$,

$$\mathbf{h}^k = \tanh(\mathbf{b}^k + W^k \mathbf{h}^{k-1}) \tag{4.1}$$

with parameters $\mathbf{b}^k$ (a vector of offsets) and W^k (a matrix of weights). The tanh is applied element-wise and can be replaced by $\text{sigm}(u) = 1/(1+e^{-u}) = \frac{1}{2}(\tanh(u)+1)$ or other saturating non-linearities. The top layer output $\mathbf{h}^\ell$ is used for making a prediction and is combined with a supervised target y into a loss function $L(\mathbf{h}^\ell, y)$, typically convex in $\mathbf{b}^\ell + W^\ell \mathbf{h}^{\ell-1}$. The output layer might have a non-linearity different from the one used in other layers, e.g., the softmax

$$\mathbf{h}_i^\ell = \frac{e^{\mathbf{b}_i^\ell + W_i^\ell \mathbf{h}^{\ell-1}}}{\sum_j e^{\mathbf{b}_j^\ell + W_j^\ell \mathbf{h}^{\ell-1}}} \tag{4.2}$$

where W_i^ℓ is the ith row of W^ℓ, $\mathbf{h}_i^\ell$ is positive and $\sum_i \mathbf{h}_i^\ell = 1$. The softmax output $\mathbf{h}_i^\ell$ can be used as estimator of $P(Y = i|\mathbf{x})$, with the

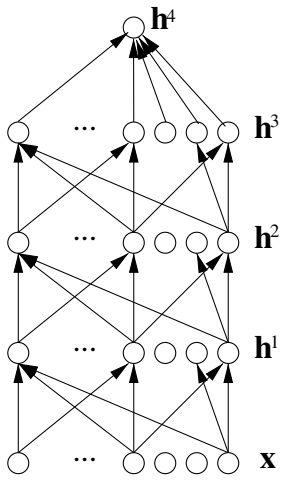

Fig. 4.1 Multi-layer neural network, typically used in supervised learning to make a prediction or classification, through a series of layers, each of which combines an affine operation and a non-linearity. Deterministic transformations are computed in a feedforward way from the input $\mathbf{x}$, through the hidden layers $\mathbf{h}^k$, to the network output $\mathbf{h}^\ell$, which gets compared with a label y to obtain the loss $L(\mathbf{h}^\ell, y)$ to be minimized.

interpretation that Y is the class associated with input pattern $\mathbf{x}$. In this case one often uses the negative conditional log-likelihood $L(\mathbf{h}^\ell, y) = -\log P(Y = y|\mathbf{x}) = -\log \mathbf{h}^\ell_y$ as a loss, whose expected value over $(\mathbf{x}, y)$ pairs is to be minimized.

4.2 The Challenge of Training Deep Neural Networks

After having motivated the need for deep architectures that are non-local estimators, we now turn to the difficult problem of training them. Experimental evidence suggests that training deep architectures is more difficult than training shallow architectures [17, 50].

Until 2006, deep architectures have not been discussed much in the machine learning literature, because of poor training and generalization errors generally obtained [17] using the standard random initialization of the parameters. Note that deep *convolutional neural networks* [104, 101, 175, 153] were found easier to train, as discussed in Section 4.5, for reasons that have yet to be really clarified.

Many unreported negative observations as well as the experimental results in [17, 50] suggest that gradient-based training of deep supervised multi-layer neural networks (starting from random initialization) gets stuck in "apparent local minima or plateaus",[1] and that as the architecture gets deeper, it becomes more difficult to obtain good generalization. When starting from random initialization, the solutions obtained with deeper neural networks appear to correspond to poor solutions that perform worse than the solutions obtained for networks with 1 or 2 hidden layers [17, 98]. This happens even though $k+1$-layer nets can easily represent what a k-layer net can represent (without much added capacity), whereas the converse is not true. However, it was discovered [73] that much better results could be achieved when pre-training each layer with an unsupervised learning algorithm, one layer after the other, starting with the first layer (that directly takes in input the observed $\mathbf{x}$). The initial experiments used the RBM generative model for each layer [73], and were followed by experiments yielding similar results using variations of auto-encoders for training each layer [17, 153, 195]. Most of these papers exploit the idea of greedy layer-wise unsupervised learning (developed in more detail in the next section): first train the lower layer with an unsupervised learning algorithm (such as one for the RBM or some auto-encoder), giving rise to an initial set of parameter values for the first layer of a neural network. Then use the output of the first layer (a new representation for the raw input) as input for another layer, and similarly initialize that layer with an unsupervised learning algorithm. After having thus initialized a number of layers, the whole neural network can be fine-tuned with respect to a supervised training criterion as usual. The advantage of unsupervised pre-training versus random initialization was clearly demonstrated in several statistical comparisons [17, 50, 98, 99]. What principles might explain the improvement in classification error observed in the literature when using unsupervised pre-training? One clue may help to identify the principles behind the success of some training algorithms for deep architectures, and it comes from algorithms that

[1] We call them apparent local minima in the sense that the gradient descent learning trajectory is stuck there, which does not completely rule out that more powerful optimizers could not find significantly better solutions far from these.

exploit neither RBMs nor auto-encoders [131, 202]. What these algorithms have in common with the training algorithms based on RBMs and auto-encoders is *layer-local unsupervised criteria*, i.e., the idea that injecting an *unsupervised training signal at each layer* may help to guide the parameters of that layer towards better regions in parameter space. In [202], the neural networks are trained using pairs of examples $(\mathbf{x}, \tilde{\mathbf{x}})$, which are either supposed to be "neighbors" (or of the same class) or not. Consider $\mathbf{h}^k(\mathbf{x})$ the level-k representation of $\mathbf{x}$ in the model. A local training criterion is defined at each layer that pushes the intermediate representations $\mathbf{h}^k(\mathbf{x})$ and $\mathbf{h}^k(\tilde{\mathbf{x}})$ either towards each other or away from each other, according to whether $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are supposed to be neighbors or not (e.g., k-nearest neighbors in input space). The same criterion had already been used successfully to learn a low-dimensional embedding with an unsupervised manifold learning algorithm [59] but is here [202] applied at one or more intermediate layer of the neural network. Following the idea of slow feature analysis [23, 131, 204] exploit the temporal constancy of high-level abstraction to provide an unsupervised guide to intermediate layers: successive frames are likely to contain the same object.

Clearly, test errors can be significantly improved with these techniques, at least for the types of tasks studied, but why? One basic question to ask is whether the improvement is basically due to better optimization or to better regularization. As discussed below, the answer may not fit the usual definition of optimization and regularization.

In some experiments [17, 98] it is clear that one can get training classification error down to zero even with a deep neural network that has no unsupervised pre-training, pointing more in the direction of a regularization effect than an optimization effect. Experiments in [50] also give evidence in the same direction: for the same training error (at different points during training), test error is systematically lower with unsupervised pre-training. As discussed in [50], unsupervised pre-training can be seen as a form of regularizer (and prior): unsupervised pre-training amounts to a constraint on the region in parameter space where a solution is allowed. The constraint forces solutions "near"[2]

[2] In the same basin of attraction of the gradient descent procedure.

ones that correspond to the unsupervised training, i.e., hopefully corresponding to solutions capturing significant statistical structure in the input. On the other hand, other experiments [17, 98] suggest that poor tuning of the lower layers might be responsible for the worse results without pre-training: when the top hidden layer is constrained (forced to be small) the *deep networks with random initialization (no unsupervised pre-training) do poorly on both training and test sets*, and much worse than pre-trained networks. In the experiments mentioned earlier where training error goes to zero, it was always the case that the number of hidden units in each layer (a hyper-parameter) was allowed to be as large as necessary (to minimize error on a validation set). The explanatory hypothesis proposed in [17, 98] is that when the top hidden layer is unconstrained, the top two layers (corresponding to a regular 1-hidden-layer neural net) are sufficient to fit the training set, using as input the representation computed by the lower layers, even if that representation is poor. On the other hand, with unsupervised pre-training, the lower layers are 'better optimized', and a smaller top layer suffices to get a low training error but also yields better generalization. Other experiments described in [50] are also consistent with the explanation that with random parameter initialization, the lower layers (closer to the input layer) are poorly trained. These experiments show that the effect of unsupervised pre-training is most marked for the lower layers of a deep architecture.

We know from experience that a two-layer network (one hidden layer) can be well trained in general, and that from the point of view of the top two layers in a deep network, they form a shallow network whose input is the output of the lower layers. Optimizing the last layer of a deep neural network is a convex optimization problem for the training criteria commonly used. Optimizing the last two layers, although not convex, is known to be much easier than optimizing a deep network (in fact when the number of hidden units goes to infinity, the training criterion of a two-layer network can be cast as convex [18]).

If there are enough hidden units (i.e., enough capacity) in the top hidden layer, training error can be brought very low even when the lower layers are not properly trained (as long as they preserve most of the information about the raw input), but this may bring worse

generalization than shallow neural networks. When training error is low and test error is high, we usually call the phenomenon overfitting. Since unsupervised pre-training brings test error down, that would point to it as a kind of data-dependent regularizer. Other strong evidence has been presented suggesting that unsupervised pre-training acts like a regularizer [50]: in particular, when there is not enough capacity, unsupervised pre-training tends to hurt generalization, and when the training set size is "small" (e.g., MNIST, with less than hundred thousand examples), although unsupervised pre-training brings improved test error, it tends to produce larger training error.

On the other hand, for much larger training sets, with better initialization of the lower hidden layers, both training and generalization error can be made significantly lower when using unsupervised pre-training (see Figure 4.2 and discussion below). We hypothesize that in a well-trained deep neural network, the hidden layers form a "good" representation of the data, which helps to make good predictions. When the lower layers are poorly initialized, these deterministic and continuous representations generally keep most of the information about the input, but these representations might scramble the input and hurt rather than help the top layers to perform classifications that generalize well.

According to this hypothesis, although replacing the top two layers of a deep neural network by convex machinery such as a Gaussian process or an SVM can yield some improvements [19], especially on the training error, it would not help much in terms of generalization if the lower layers have not been sufficiently optimized, i.e., if a good representation of the raw input has not been discovered.

Hence, one hypothesis is that unsupervised pre-training helps generalization by allowing for a 'better' tuning of lower layers of a deep architecture. Although training error can be reduced either by exploiting only the top layers ability to fit the training examples, better generalization is achieved when all the layers are tuned appropriately. Another source of better generalization could come from a form of regularization: with unsupervised pre-training, the lower layers are constrained to capture regularities of the input distribution. Consider random input-output pairs (X, Y). Such regularization is similar to the hypothesized effect of unlabeled examples in semi-supervised learning [100] or the

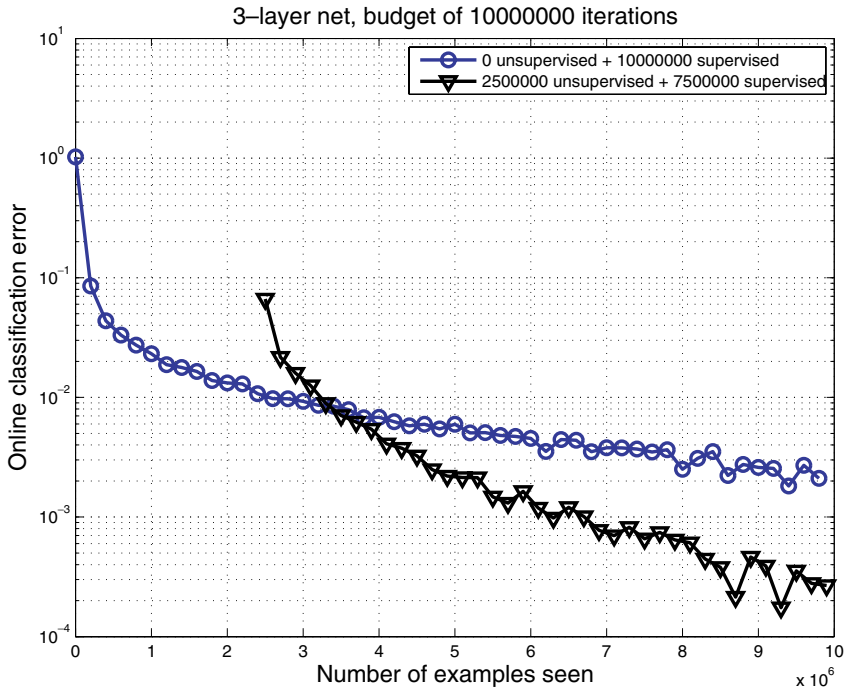

Fig. 4.2 Deep architecture trained online with 10 million examples of digit images, either with pre-training (triangles) or without (circles). The classification error shown (vertical axis, log-scale) is computed online on the next 1000 examples, plotted against the number of examples seen from the beginning. The first 2.5 million examples are used for unsupervised pre-training (of a stack of denoising auto-encoders). The oscillations near the end are because the error rate is too close to 0, making the sampling variations appear large on the log-scale. Whereas with a very large training set regularization effects should dissipate, one can see that without pre-training, training converges to a poorer apparent local minimum: unsupervised pre-training helps to find a better minimum of the online error. Experiments were performed by Dumitru Erhan.

regularization effect achieved by maximizing the likelihood of $P(X,Y)$ (generative models) vs $P(Y|X)$ (discriminant models) [118, 137]. If the true $P(X)$ and $P(Y|X)$ are unrelated as functions of X (e.g., chosen independently, so that learning about one does not inform us of the other), then unsupervised learning of $P(X)$ is not going to help learning $P(Y|X)$. But if they are related,[3] and if the same parameters are

[3] For example, the MNIST digit images form rather well-separated clusters, especially when learning good representations, even unsupervised [192], so that the decision surfaces can be guessed reasonably well even before seeing any label.

involved in estimating $P(X)$ and $P(Y|X)$,[4] then each (X, Y) pair brings information on $P(Y|X)$ not only in the usual way but also through $P(X)$. For example, in a Deep Belief Net, both distributions share essentially the same parameters, so the parameters involved in estimating $P(Y|X)$ benefit from a form of data-dependent regularization: they have to agree to some extent with $P(Y|X)$ as well as with $P(X)$.

Let us return to the optimization versus regularization explanation of the better results obtained with unsupervised pre-training. Note how one should be careful when using the word 'optimization' here. We do not have an optimization difficulty in the usual sense of the word. Indeed, from the point of view of the whole network, there is no difficulty since one can drive training error very low, by relying mostly on the top two layers. However, if one considers the problem of tuning the lower layers (while keeping small either the number of hidden units of the penultimate layer (i.e., top hidden layer) or the magnitude of the weights of the top two layers), then one can maybe talk about an optimization difficulty. One way to reconcile the optimization and regularization viewpoints might be to consider the truly online setting (where examples come from an infinite stream and one does not cycle back through a training set). In that case, online gradient descent is performing a stochastic optimization of the generalization error. If the effect of unsupervised pre-training was purely one of regularization, one would expect that with a virtually infinite training set, online error with or without pre-training would converge to the same level. On the other hand, if the explanatory hypothesis presented here is correct, we would expect that unsupervised pre-training would bring clear benefits even in the online setting. To explore that question, we have used the 'infinite MNIST' dataset [120], i.e., a virtually infinite stream of MNIST-like digit images (obtained by random translations, rotations, scaling, etc. defined in [176]). As illustrated in Figure 4.2, a 3-hidden layer neural network trained online converges to significantly lower error when it is pre-trained (as a Stacked Denoising Auto-Encoder, see Section 7.2). The figure shows progress with the online error (on the next 1000

[4] For example, all the lower layers of a multi-layer neural net estimating $P(Y|X)$ can be initialized with the parameters from a Deep Belief Net estimating $P(X)$.

examples), an unbiased Monte-Carlo estimate of generalization error. The first 2.5 million updates are used for unsupervised pre-training. The figure strongly suggests that unsupervised pre-training converges to a lower error, i.e., that it acts not only as a regularizer but also to find better minima of the optimized criterion. In spite of appearances, this does not contradict the regularization hypothesis: because of local minima, the regularization effect persists even as the number of examples goes to infinity. The flip side of this interpretation is that once the dynamics are trapped near some apparent local minimum, more labeled examples do not provide a lot more new information.

To explain that lower layers would be more difficult to optimize, the above clues suggest that the gradient propagated backwards into the lower layer might not be sufficient to move the parameters into regions corresponding to good solutions. According to that hypothesis, the optimization with respect to the lower level parameters gets stuck in a poor apparent local minimum or plateau (i.e., small gradient). Since gradient-based training of the top layers works reasonably well, it would mean that the gradient becomes less informative about the required changes in the parameters as we move back towards the lower layers, or that the error function becomes too ill-conditioned for gradient descent to escape these apparent local minima. As argued in Section 4.5, this might be connected with the observation that deep convolutional neural networks are easier to train, maybe because they have a very special sparse connectivity in each layer. There might also be a link between this difficulty in exploiting the gradient in deep networks and the difficulty in training recurrent neural networks through long sequences, analyzed in [22, 81, 119]. A recurrent neural network can be "unfolded in time" by considering the output of each neuron at different time steps as different variables, making the unfolded network over a long input sequence a very deep architecture. In recurrent neural networks, the training difficulty can be traced to a vanishing (or sometimes exploding) gradient propagated through many non-linearities. There is an additional difficulty in the case of recurrent neural networks, due to a mismatch between short-term (i.e., shorter paths in unfolded graph of computations) and long-term components of the gradient (associated with longer paths in that graph).

4.3 Unsupervised Learning for Deep Architectures

As we have seen above, layer-wise unsupervised learning has been a crucial component of all the successful learning algorithms for deep architectures up to now. If gradients of a criterion defined at the output layer become less useful as they are propagated backwards to lower layers, it is reasonable to believe that an unsupervised learning criterion defined at the level of a single layer could be used to move its parameters in a favorable direction. It would be reasonable to expect this if the single-layer learning algorithm discovered a representation that captures statistical regularities of the layer's input. PCA and the standard variants of ICA requiring as many causes as signals seem inappropriate because they generally do not make sense in the so-called *overcomplete case*, where the number of outputs of the layer is greater than the number of its inputs. This suggests looking in the direction of extensions of ICA to deal with the overcomplete case [78, 87, 115, 184], as well as algorithms related to PCA and ICA, such as auto-encoders and RBMs, which can be applied in the overcomplete case. Indeed, experiments performed with these one-layer unsupervised learning algorithms in the context of a multi-layer system confirm this idea [17, 73, 153]. Furthermore, stacking linear projections (e.g., two layers of PCA) is still a linear transformation, i.e., not building deeper architectures.

In addition to the motivation that unsupervised learning could help reduce the dependency on the unreliable update direction given by the gradient of a supervised criterion, we have already introduced another motivation for using unsupervised learning at each level of a deep architecture. It could be a way to naturally decompose the problem into sub-problems associated with different levels of abstraction. We know that unsupervised learning algorithms can extract salient information about the input distribution. This information can be captured in a distributed representation, i.e., a set of features which encode the salient factors of variation in the input. A one-layer unsupervised learning algorithm could extract such salient features, but because of the limited capacity of that layer, the features extracted on the first level of the architecture can be seen as *low-level features*. It is conceivable that learning a second layer based on the same principle but taking as input

the features learned with the first layer could extract slightly *higher-level features*. In this way, one could imagine that higher-level abstractions that characterize the input could emerge. Note how in this process all learning could remain local to each layer, therefore side-stepping the issue of gradient diffusion that might be hurting gradient-based learning of deep neural networks, when we try to optimize a single global criterion. This motivates the next section, where we discuss deep generative architectures and introduce Deep Belief Networks formally.

4.4 Deep Generative Architectures

Besides being useful for pre-training a supervised predictor, unsupervised learning in deep architectures can be of interest to learn a distribution and generate samples from it. Generative models can often be represented as graphical models [91]: these are visualized as graphs in which nodes represent random variables and arcs say something about the type of dependency existing between the random variables. The joint distribution of all the variables can be written in terms of products involving only a node and its neighbors in the graph. With directed arcs (defining parenthood), a node is conditionally independent of its ancestors, given its parents. Some of the random variables in a graphical model can be observed, and others cannot (called hidden variables). Sigmoid belief networks are generative multi-layer neural networks that were proposed and studied before 2006, and trained using variational approximations [42, 72, 164, 189]. In a sigmoid belief network, the units (typically binary random variables) in each layer are independent given the values of the units in the layer above, as illustrated in Figure 4.3. The typical parametrization of these conditional distributions (going downwards instead of upwards in ordinary neural nets) is similar to the neuron activation equation of Equation (4.1):

$$P(\mathbf{h}_i^k = 1|\mathbf{h}^{k+1}) = \text{sigm}(\mathbf{b}_i^k + \sum_j W_{i,j}^{k+1}\mathbf{h}_j^{k+1}) \tag{4.3}$$

where $\mathbf{h}_i^k$ is the binary activation of hidden node i in layer k, $\mathbf{h}^k$ is the vector $(\mathbf{h}_1^k, \mathbf{h}_2^k, \ldots)$, and we denote the input vector $\mathbf{x} = \mathbf{h}^0$. Note how the notation $P(\ldots)$ always represents a probability distribution associated with our model, whereas $\hat{P}$ is the training distribution (the

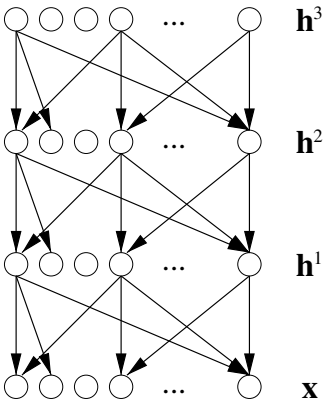

Fig. 4.3 Example of a generative multi-layer neural network, here a sigmoid belief network, represented as a directed graphical model (with one node per random variable, and directed arcs indicating direct dependence). The observed data is $\mathbf{x}$ and the hidden factors at level k are the elements of vector $\mathbf{h}^k$. The top layer $\mathbf{h}^3$ has a factorized prior.

empirical distribution of the training set, or the generating distribution for our training examples). The bottom layer generates a vector $\mathbf{x}$ in the input space, and we would like the model to give high probability to the training data. Considering multiple levels, the generative model is thus decomposed as follows:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = P(\mathbf{h}^\ell) \left(\prod_{k=1}^{\ell-1} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x}|\mathbf{h}^1) \qquad (4.4)$$

and marginalization yields $P(\mathbf{x})$, but this is intractable in practice except for tiny models. In a sigmoid belief network, the top level prior $P(\mathbf{h}^\ell)$ is generally chosen to be factorized, i.e., very simple: $P(\mathbf{h}^\ell) = \prod_i P(\mathbf{h}_i^\ell)$, and a single Bernoulli parameter is required for each $P(\mathbf{h}_i^\ell = 1)$ in the case of binary units.

Deep Belief Networks are similar to sigmoid belief networks, but with a slightly different parametrization for the top two layers, as illustrated in Figure 4.4:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell) \left(\prod_{k=1}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x}|\mathbf{h}^1). \qquad (4.5)$$

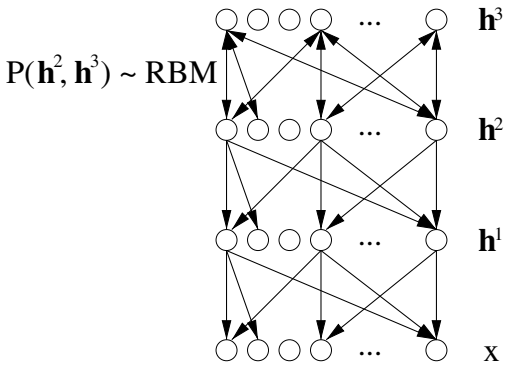

Fig. 4.4 Graphical model of a Deep Belief Network with observed vector $\mathbf{x}$ and hidden layers $\mathbf{h}^1, \mathbf{h}^2$ and $\mathbf{h}^3$. Notation is as in Figure 4.3. The structure is similar to a sigmoid belief network, except for the top two layers. Instead of having a factorized prior for $P(\mathbf{h}^3)$, the joint of the top two layers, $P(\mathbf{h}^2, \mathbf{h}^3)$, is a Restricted Boltzmann Machine. The model is mixed, with double arrows on the arcs between the top two layers because an RBM is an undirected graphical model rather than a directed one.

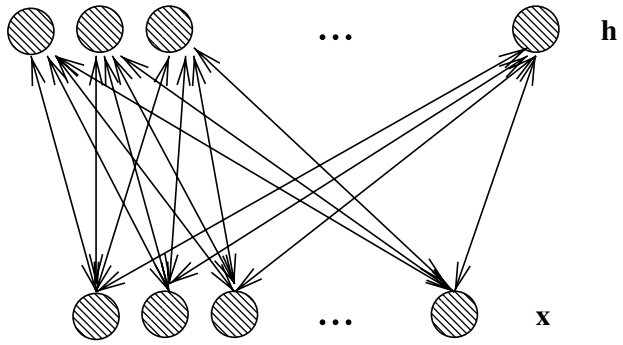

Fig. 4.5 Undirected graphical model of a Restricted Boltzmann Machine (RBM). There are no links between units of the same layer, only between input (or visible) units $\mathbf{x}_j$ and hidden units $\mathbf{h}_i$, making the conditionals $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ factorize conveniently.

The joint distribution of the top two layers is a Restricted Boltzmann Machine (RBM),

$$P(\mathbf{h}^{\ell-1}, \mathbf{h}^{\ell}) \propto e^{\mathbf{b}'\mathbf{h}^{\ell-1} + \mathbf{c}'\mathbf{h}^{\ell} + \mathbf{h}^{\ell\prime} W \mathbf{h}^{\ell-1}} \tag{4.6}$$

illustrated in Figure 4.5, and whose inference and training algorithms are described in more detail in Sections 5.3 and 5.4, respectively. This apparently slight change from sigmoidal belief networks to DBNs comes

with a different learning algorithm, which exploits the notion of training greedily one layer at a time, building up gradually more abstract representations of the raw input into the posteriors $P(\mathbf{h}^k|\mathbf{x})$. A detailed description of RBMs and of the greedy layer-wise training algorithms for deep architectures follows in Sections 5 and 6.

4.5 Convolutional Neural Networks

Although deep supervised neural networks were generally found too difficult to train before the use of unsupervised pre-training, there is one notable exception: convolutional neural networks. Convolutional nets were inspired by the visual system's structure, and in particular by the models of it proposed by [83]. The first computational models based on these local connectivities between neurons and on hierarchically organized transformations of the image are found in Fukushima's Neocognitron [54]. As he recognized, when neurons with the same parameters are applied on patches of the previous layer at different locations, a form of translational invariance is obtained. Later, LeCun and collaborators, following up on this idea, designed and trained convolutional networks using the error gradient, obtaining state-of-the-art performance [101, 104] on several pattern recognition tasks. Modern understanding of the physiology of the visual system is consistent with the processing style found in convolutional networks [173], at least for the quick recognition of objects, i.e., without the benefit of attention and top-down feedback connections. To this day, pattern recognition systems based on convolutional neural networks are among the best performing systems. This has been shown clearly for handwritten character recognition [101], which has served as a machine learning benchmark for many years.[5]

Concerning our discussion of training deep architectures, the example of convolutional neural networks [101, 104, 153, 175] is interesting because they typically have five, six or seven layers, a number of layers which makes fully connected neural networks almost impossible to train properly when initialized randomly. What is particular in their

[5] Maybe too many years? It is good that the field is moving towards more ambitious benchmarks, such as those introduced by [108, 99].

architecture that might explain their good generalization performance in vision tasks?

LeCun's convolutional neural networks are organized in layers of two types: convolutional layers and subsampling layers. Each layer has a *topographic structure*, i.e., each neuron is associated with a fixed two-dimensional position that corresponds to a location in the input image, along with a receptive field (the region of the input image that influences the response of the neuron). At each location of each layer, there are a number of different neurons, each with its set of input weights, associated with neurons in a rectangular patch in the previous layer. The same set of weights, but a different input rectangular patch, are associated with neurons at different locations.

One untested hypothesis is that the small fan-in of these neurons (few inputs per neuron) helps gradients to propagate through so many layers without diffusing so much as to become useless. Note that this alone would not suffice to explain the success of convolutional networks, since random sparse connectivity is not enough to yield good results in deep neural networks. However, an effect of the fan-in would be consistent with the idea that gradients propagated through many paths gradually become too diffuse, i.e., the credit or blame for the output error is distributed too widely and thinly. Another hypothesis (which does not necessarily exclude the first) is that the hierarchical local connectivity structure is a very strong prior that is particularly appropriate for vision tasks, and sets the parameters of the whole network in a favorable region (with all non-connections corresponding to zero weight) from which gradient-based optimization works well. The fact is that even with *random weights* in the first layers, a convolutional neural network performs well [151], i.e., better than a trained fully connected neural network but worse than a fully optimized convolutional neural network.

Very recently, the convolutional structure has been imported into RBMs [45] and DBNs [111]. An important innovation in [111] is the design of a generative version of the pooling / subsampling units, which worked beautifully in the experiments reported, yielding state-of-the-art results not only on MNIST digits but also on the Caltech-101 object classification benchmark. In addition, visualizing the features obtained

at each level (the patterns most liked by hidden units) clearly confirms the notion of multiple levels of composition which motivated deep architectures in the first place, moving up from edges to object parts to objects in a natural way.

4.6 Auto-Encoders

Some of the deep architectures discussed below (Deep Belief Nets and Stacked Auto-Encoders) exploit as component or monitoring device a particular type of neural network: the auto-encoder, also called auto-associator, or Diabolo network [25, 79, 90, 156, 172]. There are also connections between the auto-encoder and RBMs discussed in Section 5.4.3, showing that auto-encoder training approximates RBM training by Contrastive Divergence. Because training an auto-encoder seems easier than training an RBM, they have been used as building blocks to train deep networks, where each level is associated with an auto-encoder that can be trained separately [17, 99, 153, 195].

An auto-encoder is trained to encode the input $\mathbf{x}$ into some representation $\mathbf{c}(\mathbf{x})$ so that the input can be reconstructed from that representation. Hence the target output of the auto-encoder is the auto-encoder input itself. If there is one linear hidden layer and the mean squared error criterion is used to train the network, then the k hidden units learn to project the input in the span of the first k principal components of the data [25]. If the hidden layer is non-linear, the auto-encoder behaves differently from PCA, with the ability to capture multi-modal aspects of the input distribution [90]. The formulation that we prefer generalizes the mean squared error criterion to the minimization of the negative log-likelihood of the reconstruction, given the encoding $\mathbf{c}(\mathbf{x})$:

$$RE = -\log P(\mathbf{x}|\mathbf{c}(\mathbf{x})). \tag{4.7}$$

If $\mathbf{x}|\mathbf{c}(\mathbf{x})$ is Gaussian, we recover the familiar squared error. If the inputs $\mathbf{x}_i$ are either binary or considered to be binomial probabilities, then the loss function would be

$$-\log P(\mathbf{x}|\mathbf{c}(\mathbf{x})) = -\sum_i \mathbf{x}_i \log \mathbf{f}_i(\mathbf{c}(\mathbf{x})) + (1 - \mathbf{x}_i) \log(1 - \mathbf{f}_i(\mathbf{c}(\mathbf{x}))) \tag{4.8}$$

where $\mathbf{f}(\cdot)$ is called the *decoder*, and $\mathbf{f}(\mathbf{c}(\mathbf{x}))$ is the reconstruction produced by the network, and in this case should be a vector of numbers in $(0,1)$, e.g., obtained with a sigmoid. The hope is that the code $\mathbf{c}(\mathbf{x})$ is a distributed representation that captures the main factors of variation in the data: because $\mathbf{c}(\mathbf{x})$ is viewed as a lossy compression of $\mathbf{x}$, it cannot be a good compression (with small loss) for all $\mathbf{x}$, so learning drives it to be one that is a good compression in particular for training examples, and hopefully for others as well (and that is the sense in which an auto-encoder generalizes), but not for arbitrary inputs.

One serious issue with this approach is that if there is no other constraint, then an auto-encoder with n-dimensional input and an encoding of dimension at least n could potentially just learn the identity function, for which many encodings would be useless (e.g., just copying the input). Surprisingly, experiments reported in [17] suggest that in practice, when trained with stochastic gradient descent, non-linear auto-encoders with more hidden units than inputs (called overcomplete) yield useful representations (in the sense of classification error measured on a network taking this representation in input). A simple explanation is based on the observation that stochastic gradient descent with early stopping is similar to an ℓ_2 regularization of the parameters [211, 36]. To achieve perfect reconstruction of continuous inputs, a one-hidden layer auto-encoder with non-linear hidden units needs very small weights in the first layer (to bring the non-linearity of the hidden units in their linear regime) and very large weights in the second layer. With binary inputs, very large weights are also needed to completely minimize the reconstruction error. Since the implicit or explicit regularization makes it difficult to reach large-weight solutions, the optimization algorithm finds encodings which only work well for examples similar to those in the training set, which is what we want. It means that the representation is exploiting statistical regularities present in the training set, rather than learning to replicate the identity function.

There are different ways that an auto-encoder with more hidden units than inputs could be prevented from learning the identity, and still capture something useful about the input in its hidden representation. Instead or in addition to constraining the encoder by explicit or implicit

regularization of the weights, one strategy is to add noise in the encoding. This is essentially what RBMs do, as we will see later. Another strategy, which was found very successful [46, 121, 139, 150, 152, 153], is based on a sparsity constraint on the code. Interestingly, these approaches give rise to weight vectors that match well qualitatively the observed receptive fields of neurons in V1 and V2 [110], major areas of the mammal visual system. The question of sparsity is discussed further in Section 7.1.

Whereas sparsity and regularization reduce representational capacity in order to avoid learning the identity, RBMs can have a very large capacity and still not learn the identity, because they are not (only) trying to encode the input but also to capture the statistical structure in the input, by approximately maximizing the likelihood of a generative model. There is a variant of auto-encoder which shares that property with RBMs, called *denoising auto-encoder* [195]. The denoising auto-encoder minimizes the error in reconstructing the input from a stochastically corrupted transformation of the input. It can be shown that it maximizes a lower bound on the log-likelihood of a generative model. See Section 7.2 for more details.

5

Energy-Based Models and Boltzmann Machines

Because Deep Belief Networks (DBNs) are based on Restricted Boltzmann Machines (RBMs), which are particular *energy-based models*, we introduce here the main mathematical concepts helpful to understand them, including *Contrastive Divergence* (CD).

5.1 Energy-Based Models and Products of Experts

Energy-based models associate a scalar energy to each configuration of the variables of interest [107, 106, 149]. Learning corresponds to modifying that energy function so that its shape has desirable properties. For example, we would like plausible or desirable configurations to have low energy. Energy-based probabilistic models may define a probability distribution through an energy function, as follows:

$$P(\mathbf{x}) = \frac{e^{-\text{Energy}(\mathbf{x})}}{Z}, \tag{5.1}$$

i.e., energies operate in the log-probability domain. Th above generalizes *exponential family* models [29], for which the energy function $\text{Energy}(\mathbf{x})$ has the form $\eta(\theta) \cdot \phi(x)$. We will see below that the conditional distribution of one layer given another, in the RBM, can be taken

from any of the exponential family distributions [200]. Whereas any probability distribution can be cast as an energy-based models, many more specialized distribution families, such as the exponential family, can benefit from particular inference and learning procedures. Some instead have explored rather general-purpose approaches to learning in energy-based models [84, 106, 149].

The normalizing factor Z is called the *partition function* by analogy with physical systems,

$$Z = \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})} \tag{5.2}$$

with a sum running over the input space, or an appropriate integral when $\mathbf{x}$ is continuous. Some energy-based models can be defined even when the sum or integral for Z does not exist (see Section 5.1.2).

In the *product of experts* formulation [69, 70], the energy function is a sum of terms, each one associated with an "expert" f_i:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x}), \tag{5.3}$$

i.e.,

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i e^{-f_i(\mathbf{x})}. \tag{5.4}$$

Each expert $P_i(\mathbf{x})$ can thus be seen as a detector of implausible configurations of $\mathbf{x}$, or equivalently, as enforcing constraints on $\mathbf{x}$. This is clearer if we consider the special case where $f_i(\mathbf{x})$ can only take two values, one (small) corresponding to the case where the constraint is satisfied, and one (large) corresponding to the case where it is not. [69] explains the advantages of a product of experts by opposition to a *mixture of experts* where the product of probabilities is replaced by a weighted sum of probabilities. To simplify, assume that each expert corresponds to a constraint that can either be satisfied or not. In a mixture model, the constraint associated with an expert is an indication of belonging to a region which excludes the other regions. One advantage of the product of experts formulation is therefore that the set of $f_i(\mathbf{x})$ forms a distributed representation: instead of trying to partition the space with one region per expert as in mixture models, they

partition the space according to all the possible configurations (where each expert can have its constraint violated or not). [69] proposed an algorithm for estimating the gradient of $\log P(\mathbf{x})$ in Equation (5.4) with respect to parameters associated with each expert, using the first instantiation [70] of the Contrastive Divergence algorithm (Section 5.4).

5.1.1 Introducing Hidden Variables

In many cases of interest, $\mathbf{x}$ has many component variables $\mathbf{x}_i$, and we do not observe of these components simultaneously, or we want to introduce some non-observed variables to increase the expressive power of the model. So we consider an observed part (still denoted $\mathbf{x}$ here) and a *hidden* part $\mathbf{h}$

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z} \tag{5.5}$$

and because only $\mathbf{x}$ is observed, we care about the marginal

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z}. \tag{5.6}$$

In such cases, to map this formulation to one similar to Equation (5.1), we introduce the notation (inspired from physics) of *free energy*, defined as follows:

$$P(\mathbf{x}) = \frac{e^{-\text{FreeEnergy}(\mathbf{x})}}{Z}, \tag{5.7}$$

with $Z = \sum_{\mathbf{x}} e^{-\text{FreeEnergy}(\mathbf{x})}$, i.e.,

$$\text{FreeEnergy}(\mathbf{x}) = -\log \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}. \tag{5.8}$$

So the free energy is just a marginalization of energies in the log-domain. The data log-likelihood gradient then has a particularly interesting form. Let us introduce θ to represent parameters of the model. Starting from Equation (5.7), we obtain

$$\begin{aligned} \frac{\partial \log P(\mathbf{x})}{\partial \theta} &= -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \frac{1}{Z} \sum_{\tilde{\mathbf{x}}} e^{-\text{FreeEnergy}(\tilde{\mathbf{x}})} \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta} \\ &= -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}} P(\tilde{\mathbf{x}}) \ \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta}. \end{aligned} \tag{5.9}$$

Hence the average log-likelihood gradient over the training set is

$$E_{\hat{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = -E_{\hat{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + E_P\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] \tag{5.10}$$

where expectations are over $\mathbf{x}$, with $\hat{P}$ the training set empirical distribution and E_P the expectation under the model's distribution P. Therefore, if we could sample from P and compute the free energy tractably, we would have a Monte-Carlo method to obtain a stochastic estimator of the log-likelihood gradient.

If the energy can be written as a sum of terms associated with at most one hidden unit

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i), \tag{5.11}$$

a condition satisfied in the case of the RBM, then the free energy and numerator of the likelihood can be computed tractably (even though it involves a sum with an exponential number of terms):

$$\begin{aligned}
P(\mathbf{x}) &= \frac{1}{Z} e^{-\text{FreeEnergy}(\mathbf{x})} = \frac{1}{Z} \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})} \\
&= \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x}) - \sum_i \gamma_i(\mathbf{x},\mathbf{h}_i)} \\
&= \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x})} \prod_i e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \sum_{\mathbf{h}_1} e^{-\gamma_1(\mathbf{x},\mathbf{h}_1)} \sum_{\mathbf{h}_2} e^{-\gamma_2(\mathbf{x},\mathbf{h}_2)} \cdots \sum_{\mathbf{h}_k} e^{-\gamma_k(\mathbf{x},\mathbf{h}_k)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \prod_i \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}
\end{aligned} \tag{5.12}$$

In the above, $\sum_{\mathbf{h}_i}$ is a sum over all the values that $\mathbf{h}_i$ can take (e.g., two values in the usual binomial units case); note how that sum is much easier to carry out than the sum $\sum_{\mathbf{h}}$ over all values of $\mathbf{h}$. Note that all sums can be replaced by integrals if $\mathbf{h}$ is continuous, and the same principles apply. In many cases of interest, the sum or integral (over a single hidden unit's values) is easy to compute. The numerator of the

likelihood (i.e., also the free energy) can be computed exactly in the above case, where $\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$, and we have

$$\text{FreeEnergy}(\mathbf{x}) = -\log P(\mathbf{x}) - \log Z = -\beta(\mathbf{x}) - \sum_i \log \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x}, \mathbf{h}_i)}. \tag{5.13}$$

5.1.2 Conditional Energy-Based Models

Whereas computing the partition function is difficult in general, if our ultimate goal is to make a decision concerning a variable y given a variable $\mathbf{x}$, instead of considering all configurations $(\mathbf{x}, y)$, it is enough to consider the configurations of y for each given $\mathbf{x}$. A common case is one where y can only take values in a small discrete set, i.e.,

$$P(y|\mathbf{x}) = \frac{e^{-\text{Energy}(\mathbf{x}, y)}}{\sum_y e^{-\text{Energy}(\mathbf{x}, y)}}. \tag{5.14}$$

In this case the gradient of the conditional log-likelihood with respect to parameters of the energy function can be computed efficiently. This formulation applies to a discriminant variant of the RBM called Discriminative RBM [97]. Such conditional energy-based models have also been exploited in a series of probabilistic language models based on neural networks [15, 16, 130, 169, 170, 171, 207]. That formulation (or generally when it is easy to sum or maximize over the set of values of the terms of the partition function) has been explored at length [37, 106, 107, 149, 153]. An important and interesting element in the latter work is that it shows that such energy-based models can be optimized not just with respect to log-likelihood but with respect to more general criteria whose gradient has the property of making the energy of "correct" responses decrease while making the energy of competing responses increase. These energy functions do not necessarily give rise to a probabilistic model (because the exponential of the negated energy function is not required to be integrable), but they may nonetheless give rise to a function that can be used to choose y given $\mathbf{x}$, which is often the ultimate goal in applications. Indeed when y takes a finite number of values, $P(y|\mathbf{x})$ can always be computed since the energy function needs to be normalized only over the possible values of y.

5.2 Boltzmann Machines

The Boltzmann machine is a particular type of energy-based model with hidden variables, and RBMs are special forms of Boltzmann machines in which $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ are both tractable because they factorize. In a Boltzmann machine [1, 76, 77], the energy function is a general second-order polynomial:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x} - \mathbf{x}'U\mathbf{x} - \mathbf{h}'V\mathbf{h}. \tag{5.15}$$

There are two types of parameters, which we collectively denote by θ: the offsets $\mathbf{b}_i$ and $\mathbf{c}_i$ (each associated with a single element of the vector $\mathbf{x}$ or of the vector $\mathbf{h}$), and the weights W_{ij}, U_{ij} and V_{ij} (each associated with a pair of units). Matrices U and V are assumed to be symmetric,[1] and in most models with zeros in the diagonal. Non-zeros in the diagonal can be used to obtain other variants, e.g., with Gaussian instead of binomial units [200].

Because of the quadratic interaction terms in $\mathbf{h}$, the trick to analytically compute the free energy (Equation (5.12)) cannot be applied here. However, an MCMC (Monte Carlo Markov Chain [4]) sampling procedure can be applied in order to obtain a stochastic estimator of the gradient. The gradient of the log-likelihood can be written as follows, starting from Equation (5.6):

$$\begin{aligned}
\frac{\partial \log P(\mathbf{x})}{\partial \theta} &= \frac{\partial \log \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{\partial \theta} - \frac{\partial \log \sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}}{\partial \theta} \\
&= -\frac{1}{\sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}} \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})} \frac{\partial \text{Energy}(\mathbf{x},\mathbf{h})}{\partial \theta} \\
&\quad + \frac{1}{\sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}} \sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})} \frac{\partial \text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}{\partial \theta} \\
&= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x},\mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}},\mathbf{h}} P(\tilde{\mathbf{x}},\mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}{\partial \theta}.
\end{aligned} \tag{5.16}$$

[1] For example, if U was not symmetric, the extra degrees of freedom would be wasted since $\mathbf{x}_i U_{ij} \mathbf{x}_j + \mathbf{x}_j U_{ji} \mathbf{x}_i$ can be rewritten $\mathbf{x}_i(U_{ij} + U_{ji})\mathbf{x}_j = \frac{1}{2}\mathbf{x}_i(U_{ij} + U_{ji})\mathbf{x}_j + \frac{1}{2}\mathbf{x}_j(U_{ij} + U_{ji})\mathbf{x}_i$, i.e., in a symmetric-matrix form.

Note that $(\partial \text{Energy}(\mathbf{x}, \mathbf{h})/\partial\theta)$ is easy to compute. Hence if we have a procedure to sample from $P(\mathbf{h}|\mathbf{x})$ and one to sample from $P(\mathbf{x}, \mathbf{h})$, we can obtain an unbiased stochastic estimator of the log-likelihood gradient. [1, 76, 77] introduced the following terminology: in the *positive phase*, $\mathbf{x}$ is *clamped* to the observed input vector, and we sample $\mathbf{h}$ given $\mathbf{x}$; in the *negative phase* both $\mathbf{x}$ and $\mathbf{h}$ are sampled, ideally from the model itself. In general, only approximate sampling can be achieved tractably, e.g., using an iterative procedure that constructs an MCMC. The MCMC sampling approach introduced in [1, 76, 77] is based on *Gibbs sampling* [4, 57]. Gibbs sampling of the joint of N random variables $S = (S_1, \ldots, S_N)$ is done through a sequence of N sampling sub-steps of the form

$$S_i \sim P(S_i | S_{-i} = \mathbf{s}_{-i}) \tag{5.17}$$

where S_{-i} contains the $N - 1$ other random variables in S, excluding S_i. After these N samples have been obtained, a step of the chain is completed, yielding a sample of S whose distribution converges to $P(S)$ as the number of steps goes to ∞, under some conditions. A sufficient condition for convergence of a finite-state Markov Chain is that it is aperiodic[2] and irreducible.[3]

How can we perform Gibbs sampling in a Boltzmann machine? Let $\mathbf{s} = (\mathbf{x}, \mathbf{h})$ denote all the units in the Boltzmann machine, and $\mathbf{s}_{-i}$ the set of values associated with all units except the ith one. The Boltzmann machine energy function can be rewritten by putting all the parameters in a vector $\mathbf{d}$ and a symmetric matrix A,

$$\text{Energy}(\mathbf{s}) = -\mathbf{d}'\mathbf{s} - \mathbf{s}'A\mathbf{s}. \tag{5.18}$$

Let $\mathbf{d}_{-i}$ denote the vector $\mathbf{d}$ without the element $\mathbf{d}_i$, A_{-i} the matrix A without the ith row and column, and $\mathbf{a}_{-i}$ the vector that is the ith row (or column) of A, without the ith element. Using this notation, we obtain that $P(\mathbf{s}_i|\mathbf{s}_{-i})$ can be computed and sampled from easily in a Boltzmann machine. For example, if $\mathbf{s}_i \in \{0, 1\}$ and the diagonal of A

[2] Aperiodic: no state is periodic with period $k > 1$; a state has period k if one can only return to it at times $t + k$, $t + 2k$, etc.

[3] Irreducible: one can reach any state from any state in finite time with non-zero probability.

is null:

$$
\begin{aligned}
P(\mathbf{s}_i = 1|\mathbf{s}_{-i}) &= \frac{\exp(\mathbf{d}_i + \mathbf{d}'_{-i}\mathbf{s}_{-i} + 2\mathbf{a}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i})}{\exp(\mathbf{d}_i + \mathbf{d}'_{-i}\mathbf{s}_{-i} + 2\mathbf{a}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i}) \\ &\qquad + \exp(\mathbf{d}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i})} \\
&= \frac{\exp(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i})}{\exp(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i}) + 1} = \frac{1}{1 + \exp(-\mathbf{d}_i - 2\mathbf{a}'_{-i}\mathbf{s}_{-i})} \\
&= \text{sigm}(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i}) \qquad (5.19)
\end{aligned}
$$

which is essentially the usual equation for computing a neuron's output in terms of other neurons $\mathbf{s}_{-i}$, in artificial neural networks.

Since two MCMC chains (one for the positive phase and one for the negative phase) are needed for each example $\mathbf{x}$, the computation of the gradient can be very expensive, and training time very long. This is essentially why the Boltzmann machine was replaced in the late 1980's by the back-propagation algorithm for multi-layer neural networks as the dominant learning approach. However, recent work has shown that short chains can sometimes be used successfully, and this is the principle of Contrastive Divergence, discussed in Section 5.4 to train RBMs. Note also that the negative phase chain does not have to be restarted for each new example $\mathbf{x}$ (since it does not depend on the training data), and this observation has been exploited in persistent MCMC estimators [161, 187] discussed in Section 5.4.2.

5.3 Restricted Boltzmann Machines

The *Restricted* Boltzmann Machine (RBM) is the building block of a Deep Belief Network (DBN) because it shares parametrization with individual layers of a DBN, and because efficient learning algorithms were found to train it. The undirected graphical model of an RBM is illustrated in Figure 4.5, showing that the $\mathbf{h}_i$ are independent of each other when conditioning on $\mathbf{x}$ and the $\mathbf{x}_j$ are independent of each other when conditioning on $\mathbf{h}$. In an RBM, $U = 0$ and $V = 0$ in Equation (5.15), i.e., the only interaction terms are between a hidden unit and a visible unit, but not between units of the same layer. This form of model was first introduced under the name of *Harmonium* [178], and learning algorithms (beyond the ones for Boltzmann Machines) were

discussed in [51]. Empirically demonstrated and efficient learning algorithms and variants were proposed more recently [31, 70, 200]. As a consequence of the lack of input–input and hidden-hidden interactions, the energy function is bilinear,

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x} \tag{5.20}$$

and the factorization of the free energy of the input, introduced with Equations (5.11) and (5.13) can be applied with $\beta(\mathbf{x}) = \mathbf{b}'\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h}_i) = -\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$, where W_i is the row vector corresponding to the ith row of W. Therefore the free energy of the input (i.e., its unnormalized log-probability) can be computed efficiently:

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}'\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} e^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}. \tag{5.21}$$

Using the same factorization trick (in Equation (5.12)) due to the affine form of $\text{Energy}(\mathbf{x}, \mathbf{h})$ with respect to $\mathbf{h}$, we readily obtain a tractable expression for the conditional probability $P(\mathbf{h}|\mathbf{x})$:

$$\begin{aligned} P(\mathbf{h}|\mathbf{x}) &= \frac{\exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\mathbf{h} + \mathbf{h}'W\mathbf{x})}{\sum_{\tilde{\mathbf{h}}} \exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\tilde{\mathbf{h}} + \tilde{\mathbf{h}}'W\mathbf{x})} \\ &= \frac{\prod_i \exp(\mathbf{c}_i\mathbf{h}_i + \mathbf{h}_i W_i\mathbf{x})}{\prod_i \sum_{\tilde{\mathbf{h}}_i} \exp(\mathbf{c}_i\tilde{\mathbf{h}}_i + \tilde{\mathbf{h}}_i W_i\mathbf{x})} \\ &= \prod_i \frac{\exp(\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x}))}{\sum_{\tilde{\mathbf{h}}_i} \exp(\tilde{\mathbf{h}}_i(\mathbf{c}_i + W_i\mathbf{x}))} \\ &= \prod_i P(\mathbf{h}_i|\mathbf{x}). \end{aligned}$$

In the commonly studied case where $\mathbf{h}_i \in \{0, 1\}$, we obtain the usual neuron equation for a neuron's output given its input:

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \frac{e^{\mathbf{c}_i + W_i\mathbf{x}}}{1 + e^{\mathbf{c}_i + W_i\mathbf{x}}} = \text{sigm}(\mathbf{c}_i + W_i\mathbf{x}). \tag{5.22}$$

Since $\mathbf{x}$ and $\mathbf{h}$ play a symmetric role in the energy function, a similar derivation allows to efficiently compute and sample $P(\mathbf{x}|\mathbf{h})$:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h}) \tag{5.23}$$

and in the binary case

$$P(\mathbf{x}_j = 1|\mathbf{h}) = \text{sigm}(\mathbf{b}_j + W'_{\cdot j}\mathbf{h}) \tag{5.24}$$

where $W_{\cdot j}$ is the jth column of W.

In [73], binomial input units are used to encode pixel gray levels in input images as if they were the probability of a binary event. In the case of handwritten character images this approximation works well, but in other cases it does not. Experiments showing the advantage of using Gaussian input units rather than binomial units when the inputs are continuous-valued are described in [17]. See [200] for a general formulation where $\mathbf{x}$ and $\mathbf{h}$ (given the other) can be in any of the exponential family distributions (discrete and continuous).

Although RBMs might not be able to represent efficiently some distributions that could be represented compactly with an unrestricted Boltzmann machine, RBMs can represent any discrete distribution [51, 102], if enough hidden units are used. In addition, it can be shown that unless the RBM already perfectly models the training distribution, adding a hidden unit (and properly choosing its weights and offset) can always improve the log-likelihood [102].

An RBM can also be seen as forming a multi-clustering (see Section 3.2), as illustrated in Figure 3.2. Each hidden unit creates a two-region partition of the input space (with a linear separation). When we consider the configurations of say three hidden units, there are eight corresponding possible intersections of three half-planes (by choosing each half-plane among the two half-planes associated with the linear separation performed by a hidden unit). Each of these eight intersections corresponds to a region in input space associated with the same hidden configuration (i.e., code). The binary setting of the hidden units thus identifies one region in input space. For all $\mathbf{x}$ in one of these regions, $P(\mathbf{h}|\mathbf{x})$ is maximal for the corresponding $\mathbf{h}$ configuration. Note that not all configurations of the hidden units correspond to a non-empty region in input space. As illustrated in Figure 3.2, this representation is similar to what an ensemble of two-leaf trees would create.

The sum over the exponential number of possible hidden-layer configurations of an RBM can also be seen as a particularly interesting form

of mixture, with an exponential number of components (with respect to the number of hidden units and of parameters):

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}|\mathbf{h})P(\mathbf{h}) \tag{5.25}$$

where $P(\mathbf{x}|\mathbf{h})$ is the model associated with the component indexed by configuration $\mathbf{h}$. For example, if $P(\mathbf{x}|\mathbf{h})$ is chosen to be Gaussian (see [200, 17]), this is a Gaussian mixture with 2^n components when $\mathbf{h}$ has n bits. Of course, these 2^n components cannot be tuned independently because they depend on shared parameters (the RBM parameters), and that is also the strength of the model, since it can generalize to configurations (regions of input space) for which no training example was seen. We can see that the Gaussian mean (in the Gaussian case) associated with component $\mathbf{h}$ is obtained as a linear combination $\mathbf{b} + W'\mathbf{h}$, i.e., each hidden unit bit $\mathbf{h}_i$ contributes (or not) a vector W_i in the mean.

5.3.1 Gibbs Sampling in RBMs

Sampling from an RBM is useful for several reasons. First of all it is useful in learning algorithms, to obtain an estimator of the log-likelihood gradient. Second, inspection of examples generated from the model is useful to get an idea of what the model has captured or not captured about the data distribution. Since the joint distribution of the top two layers of a DBN is an RBM, sampling from an RBM enables us to sample from a DBN, as elaborated in Section 6.1.

Gibbs sampling in fully connected Boltzmann Machines is slow because there are as many sub-steps in the Gibbs chain as there are units in the network. On the other hand, the factorization enjoyed by RBMs brings two benefits: first we do not need to sample in the positive phase because the free energy (and therefore its gradient) is computed analytically; second, the set of variables in $(\mathbf{x}, \mathbf{h})$ can be sampled in two sub-steps in each step of the Gibbs chain. First we sample $\mathbf{h}$ given $\mathbf{x}$, and then a new $\mathbf{x}$ given $\mathbf{h}$. In general product of experts models, an alternative to Gibbs sampling is hybrid Monte-Carlo [48, 136], an MCMC method involving a number of free-energy gradient computation sub-steps for each step of the Markov chain. The RBM structure

is therefore a special case of product of experts model: the ith term $\log \sum_{\mathbf{h}_i} e^{(\mathbf{c}_i + W_i \mathbf{x})\mathbf{h}_i}$ in Equation (5.21) corresponds to an expert, i.e., there is one expert per hidden neuron and one for the input offset. With that special structure, a very efficient Gibbs sampling can be performed. For k Gibbs steps, starting from a training example (i.e., sampling from $\hat{P}$):

$$\begin{aligned} \mathbf{x}_1 &\sim \hat{P}(\mathbf{x}) \\ \mathbf{h}_1 &\sim P(\mathbf{h}|\mathbf{x}_1) \\ \mathbf{x}_2 &\sim P(\mathbf{x}|\mathbf{h}_1) \\ \mathbf{h}_2 &\sim P(\mathbf{h}|\mathbf{x}_2) \\ &\vdots \\ \mathbf{x}_{k+1} &\sim P(\mathbf{x}|\mathbf{h}_k). \end{aligned} \tag{5.26}$$

It makes sense to start the chain from a training example because as the model becomes better at capturing the structure in the training data, the model distribution P and the training distribution $\hat{P}$ become more similar (having similar statistics). Note that if we started the chain from P itself, it would have converged in one step, so starting from $\hat{P}$ is a good way to ensure that only a few steps are necessary for convergence.

5.4 Contrastive Divergence

Contrastive Divergence is an approximation of the log-likelihood gradient that has been found to be a successful update rule for training RBMs [31]. A pseudo-code is shown in Algorithm 1, with the particular equations for the conditional distributions for the case of binary input and hidden units.

5.4.1 Justifying Contrastive Divergence

To obtain this algorithm, the **first approximation** we are going to make is replace the average over all possible inputs (in the second term of Equation (5.10)) by a single sample. Since we update the parameters often (e.g., with stochastic or mini-batch gradient updates after one or a few training examples), there is already some averaging going on across

Algorithm 1

`RBMupdate`$(\mathbf{x}_1, \epsilon, W, \mathbf{b}, \mathbf{c})$

This is the RBM update procedure for binomial units. It can easily adapted to other types of units.

$\mathbf{x}_1$ is a sample from the training distribution for the RBM

ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence

W is the RBM weight matrix, of dimension (number of hidden units, number of inputs)

$\mathbf{b}$ is the RBM offset vector for input units

$\mathbf{c}$ is the RBM offset vector for hidden units

Notation: $Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)$ is the vector with elements $Q(\mathbf{h}_{2i} = 1|\mathbf{x}_2)$

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i}=1|\mathbf{x}_1)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{1j})$)
- sample $\mathbf{h}_{1i} \in \{0,1\}$ from $Q(\mathbf{h}_{1i}|\mathbf{x}_1)$

end for

for all visible units j **do**
- compute $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$ (for binomial units, $\text{sigm}(\mathbf{b}_j + \sum_i W_{ij}\mathbf{h}_{1i})$)
- sample $\mathbf{x}_{2j} \in \{0,1\}$ from $P(\mathbf{x}_{2j} = 1|\mathbf{h}_1)$

end for

for all hidden units i **do**
- compute $Q(\mathbf{h}_{2i}=1|\mathbf{x}_2)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{2j})$)

end for

- $W \leftarrow W + \epsilon(\mathbf{h}_1\mathbf{x}_1' - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)\mathbf{x}_2')$
- $\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{x}_1 - \mathbf{x}_2)$
- $\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2))$

updates (which we know to work well [105]), and the extra variance introduced by taking one or a few MCMC samples instead of doing the complete sum might be partially canceled in the process of online gradient updates, over consecutive parameter updates. We introduce additional variance with this approximation of the gradient, but it does not hurt much if it is comparable or smaller than the variance due to online gradient descent.

Running a long MCMC chain is still very expensive. The idea of k-step Contrastive Divergence (CD-k) [69, 70] is simple, and involves a **second approximation**, which introduces some bias in the gradient: run the MCMC chain $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k+1}$ for only k steps *starting from the observed example* $\mathbf{x}_1 = \mathbf{x}$. The CD-$k$ update (i.e., not the log-likelihood gradient) after seeing example $\mathbf{x}$ is, therefore,

$$\Delta\theta \propto \frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial\theta} - \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial\theta} \tag{5.27}$$

where $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$ is the last sample from our Markov chain, obtained after k steps. We know that when $k \to \infty$, the bias goes away. We also know that when the model distribution is very close to the empirical distribution, i.e., $P \approx \hat{P}$, then when we start the chain from $\mathbf{x}$ (a sample from $\hat{P}$) the MCMC has already converged, and we need only one step to obtain an unbiased sample from P (although it would still be correlated with $\mathbf{x}$).

The surprising empirical result is that even $k = 1$ (CD-1) often gives good results. An extensive numerical comparison of training with CD-k versus exact log-likelihood gradient has been presented in [31]. In these experiments, taking k larger than 1 gives more precise results, although very good approximations of the solution can be obtained even with $k = 1$. Theoretical results [12] discussed in Section 5.4.3 help to understand why small values of k can work: CD-k corresponds to keeping the first k terms of a series that converges to the log-likelihood gradient.

One way to interpret Contrastive Divergence is that it is approximating the log-likelihood gradient *locally* around the training point $\mathbf{x}_1$. The stochastic reconstruction $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$ (for CD-k) has a distribution (given $\mathbf{x}_1$) which is in some sense centered around $\mathbf{x}_1$ and becomes more spread out around it as k increases, until it becomes the model distribution. The CD-k update will decrease the free energy of the training point $\mathbf{x}_1$ (which would increase its likelihood if all the other free energies were kept constant), and increase the free energy of $\tilde{\mathbf{x}}$, which is in the neighborhood of $\mathbf{x}_1$. Note that $\tilde{\mathbf{x}}$ is in the neighborhood of $\mathbf{x}_1$, but at the same time more likely to be in regions of high probability under the model (especially for k larger). As argued by [106],

what is mostly needed from the training algorithm for an energy-based model is that it makes the energy (free energy, here, to marginalize hidden variables) of observed inputs smaller, shoveling "energy" elsewhere, and most importantly in areas of low energy. The Contrastive Divergence algorithm is fueled by the *contrast* between the statistics collected when the input is a real training example and when the input is a chain sample. As further argued in the next section, one can think of the unsupervised learning problem as discovering a decision surface that can roughly separate the regions of high probability (where there are many observed training examples) from the rest. Therefore, we want to penalize the model when it generates examples on the wrong side of that divide, and a good way to identify where that divide should be moved is to compare training examples with samples from the model.

5.4.2 Alternatives to Contrastive Divergence

An exciting recent development in the research on learning algorithms for RBMs is use of a so-called persistent MCMC for the negative phase [161, 187], following an approach already introduced in [135]. The idea is simple: keep a background MCMC chain $\ldots \mathbf{x}_t \rightarrow \mathbf{h}_t \rightarrow \mathbf{x}_{t+1} \rightarrow \mathbf{h}_{t+1} \ldots$ to obtain the negative phase samples (which should be from the model). Instead of running a short chain as in CD-k, the approximation made is that we ignore the fact that parameters are changing as we move along the chain, i.e., we do not run a separate chain for each value of the parameters (as in the traditional Boltzmann Machine learning algorithm). Maybe because the parameters move slowly, the approximation works very well, usually giving rise to better log-likelihood than CD-k (experiments were against $k = 1$ and $k = 10$). The trade-off with CD-1 is that the variance is larger but the bias is smaller. Something interesting also happens [188]: the model systematically moves away from the samples obtained in the negative phase, and this interacts with the chain itself, preventing it from staying in the same region very long, substantially improving the mixing rate of the chain. This is a very desirable and unforeseen effect, which helps to explore more quickly the space of RBM configurations.

Another alternative to Contrastive Divergence is Score Matching [84, 85, 86], a general approach to train energy-based models in which the energy can be computed tractably, but not the normalization constant Z. The score function of a density $p(\mathbf{x}) = q(\mathbf{x})/Z$ is $\psi = (\partial \log p(\mathbf{x}))/\partial \mathbf{x}$, and we exploit the fact that the score function of our model does not depend on its normalization constant, i.e., $\psi = (\partial \log q(\mathbf{x}))/\partial \mathbf{x}$. The basic idea is to match the score function of the model with the score function of the empirical density. The average (under the empirical density) of the squared norm of the difference between the two score functions can be written in terms of squares of the model score function and second derivatives $(\partial^2 \log q(\mathbf{x}))/\partial \mathbf{x}^2$. Score matching has been shown to be locally consistent [84], i.e., converging if the model family matches the data generating process, and it has been used for unsupervised models of image and audio data [94].

5.4.3 Truncations of the Log-Likelihood Gradient in Gibbs-Chain Models

Here, we approach the Contrastive Divergence update rule from a different perspective, which gives rise to possible generalizations of it and links it to the reconstruction error often used to monitor its performance and that is used to optimize auto-encoders (Equation (4.7)). The inspiration for this derivation comes from [73]: first from the idea (explained in Section 8.1) that the Gibbs chain can be associated with an infinite directed graphical model (which here we associate with an expansion of the log-likelihood gradient), and second that the convergence of the chain justifies Contrastive Divergence (since the expected value of Equation (5.27) becomes equivalent to Equation (5.9) when the chain sample $\tilde{\mathbf{x}}$ comes from the model). In particular, we are interested in clarifying and understanding the bias in the Contrastive Divergence update rule, compared to using the true (intractable) gradient of the log-likelihood.

Consider a converging Markov chain $\mathbf{x}_t \Rightarrow \mathbf{h}_t \Rightarrow \mathbf{x}_{t+1} \Rightarrow \cdots$ defined by conditional distributions $P(\mathbf{h}_t|\mathbf{x}_t)$ and $P(\mathbf{x}_{t+1}|\mathbf{h}_t)$, with $\mathbf{x}_1$ sampled from the training data empirical distribution. The following theorem,

demonstrated by [12], shows how one can expand the log-likelihood gradient for any $t \geq 1$.

Theorem 5.1. Consider the converging Gibbs chain $\mathbf{x}_1 \Rightarrow \mathbf{h}_1 \Rightarrow \mathbf{x}_2 \Rightarrow \mathbf{h}_2 \cdots$ starting at data point $\mathbf{x}_1$. The log-likelihood gradient can be written

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} = -\frac{\partial \text{FreeEnergy}(\mathbf{x}_1)}{\partial \theta} + E\left[\frac{\partial \text{FreeEnergy}(\mathbf{x}_t)}{\partial \theta}\right] + E\left[\frac{\partial \log P(\mathbf{x}_t)}{\partial \theta}\right] \tag{5.28}$$

and the final term converges to zero as t goes to infinity.

Since the final term becomes small as t increases, that justifies truncating the chain to k steps in the Markov chain, using the approximation

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} \simeq -\frac{\partial \text{FreeEnergy}(\mathbf{x}_1)}{\partial \theta} + E\left[\frac{\partial \text{FreeEnergy}(\mathbf{x}_{k+1})}{\partial \theta}\right]$$

which is exactly the CD-k update (Equation (5.27)) when we replace the expectation with a single sample $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$. This tells us that the bias of CD-k is $E[(\partial \log P(\mathbf{x}_{k+1}))/\partial\theta]$. Experiments and theory support the idea that CD-k yields better and faster convergence (in terms of number of iterations) than CD-$(k-1)$, due to smaller bias (though the computational overhead might not always be worth it). However, although experiments show that the CD-k bias can indeed be large when k is small, empirically the update rule of CD-k still mostly moves the model's parameters in the same quadrant as log-likelihood gradient [12]. This is in agreement with the good results can be obtained even with $k = 1$. An intuitive picture that may help to understand the phenomenon is the following: when the input example $\mathbf{x}_1$ is used to initialize the chain, even the first Markov chain step (to $\mathbf{x}_2$) tends to be in the right direction compared to $\mathbf{x}_1$, i.e., roughly going down the energy landscape from $\mathbf{x}_1$. Since the gradient depends on the change between $\mathbf{x}_2$ and $\mathbf{x}_1$, we tend to get the direction of the gradient right.

So CD-1 corresponds to truncating the chain after two samples (one from $\mathbf{h}_1|\mathbf{x}_1$, and one from $\mathbf{x}_2|\mathbf{h}_1$). What about stopping after the first one (i.e., $\mathbf{h}_1|\mathbf{x}_1$)? It can be analyzed from the following log-likelihood gradient expansion [12]:

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} = E\left[\frac{\partial \log P(\mathbf{x}_1|\mathbf{h}_1)}{\partial \theta}\right] - E\left[\frac{\partial \log P(\mathbf{h}_1)}{\partial \theta}\right]. \quad (5.29)$$

Let us consider a mean-field approximation of the first expectation, in which instead of the average over all $\mathbf{h}_1$ configurations according to $P(\mathbf{h}_1|\mathbf{x}_1)$ one replaces $\mathbf{h}_1$ by its average configuration $\widehat{\mathbf{h}}_1 = E[\mathbf{h}_1|\mathbf{x}_1]$, yielding:

$$E\left[\frac{\partial \log P(\mathbf{x}_1|\mathbf{h}_1)}{\partial \theta}\right] \simeq \frac{\partial \log P(\mathbf{x}_1|\widehat{\mathbf{h}}_1)}{\partial \theta}. \quad (5.30)$$

If, as in CD, we then ignore the second expectation in Equation (5.29) (incurring an additional bias in the estimation of the log-likelihood gradient), we then obtain the right-hand side of Equation (5.30) as an update direction, which is minus the gradient of the *reconstruction error*,

$$-\log P(\mathbf{x}_1|\widehat{\mathbf{h}}_1)$$

typically used to train auto-encoders (see Equation (4.7) with $\mathbf{c}(\mathbf{x}) = E[\mathbf{h}|\mathbf{x}]$).[4]

So we have found that the truncation of the Gibbs chain gives rise to first approximation (one sample) to roughly reconstruction error (through a biased mean-field approximation), with slightly better approximation (two samples) to CD-1 (approximating the expectation by a sample), and with more terms to CD-k (still approximating expectations by samples). Note that reconstruction error is deterministically computed and is correlated with log-likelihood, which is why it has been used to track progress when training RBMs with CD.

[4] It is debatable whether or not one would take into account the fact that $\widehat{\mathbf{h}}_1$ depends on θ when computing the gradient in the mean-field approximation of Equation (5.30), but it must be the case to draw a direct link with auto-encoders.

5.4.4 Model Samples Are Negative Examples

Here, we argue that training an energy-based model can be achieved by solving a series of classification problems in which one tries to discriminate training examples from samples generated by the model. In the Boltzmann machine learning algorithms, as well as in Contrastive Divergence, an important element is the ability to *sample from the model*, maybe approximately. An elegant way to understand the value of these samples in improving the log-likelihood was introduced in [201], using a connection with boosting. We start by explaining the idea informally and then formalize it, justifying algorithms based on training the generative model with a classification criterion *separating model samples from training examples*. The maximum likelihood criterion wants the likelihood to be high on the training examples and low elsewhere. If we already have a model and we want to increase its likelihood, the contrast between where the model puts high probability (represented by samples) and where the training examples are indicates how to change the model. If we were able to approximately separate training examples from model samples with a decision surface, we could increase likelihood by reducing the value of the energy function on one side of the decision surface (the side where there are more training examples) and increasing it on the other side (the side where there are more samples from the model). Mathematically, consider the gradient of the log-likelihood with respect to the parameters of the FreeEnergy($\mathbf{x}$) (or Energy($\mathbf{x}$) if we do not introduce explicit hidden variables), given in Equation (5.10). Now consider a highly regularized two-class probabilistic classifier that will attempt to separate training samples of $\hat{P}(\mathbf{x})$ from model samples of $P(\mathbf{x})$, and which is only able to produce an output probability $q(\mathbf{x}) = P(y = 1|\mathbf{x})$ barely different from $\frac{1}{2}$ (hopefully on the right side more often than not). Let $q(\mathbf{x}) = \text{sigm}(-a(\mathbf{x}))$, i.e., $-a(\mathbf{x})$ is the discriminant function or an unnormalized conditional log-probability, just like the free energy. Let $\tilde{P}$ denote the empirical distribution over $(\mathbf{x}, y)$ pairs, and $\tilde{P}_i$ the distribution over $\mathbf{x}$ when $y = i$. Assume that $\tilde{P}(y = 1) = \tilde{P}(y = 0) = 1/2$, so that $\forall f,\ E_{\tilde{P}}[f(\mathbf{x}, y)] = E_{\tilde{P}_1}[f(\mathbf{x}, 1)]\tilde{P}(y = 1) + E_{\tilde{P}_0}[f(\mathbf{x}, 0)]\tilde{P}(y = 0) = \frac{1}{2}(E_{\tilde{P}_1}[f(\mathbf{x}, 1)] + E_{\tilde{P}_0}[f(\mathbf{x}, 0)])$. Using this, the average conditional

log-likelihood gradient for this probabilistic classifier is written

$$\begin{aligned} E_{\tilde{P}}\left[\frac{\partial \log P(y|\mathbf{x})}{\partial \theta}\right] &= E_{\tilde{P}}\left[\frac{\partial (y \log q(\mathbf{x}) + (1-y)\log(1-q(\mathbf{x})))}{\partial \theta}\right] \\ &= \frac{1}{2}\left(E_{\tilde{P}_1}\left[(q(\mathbf{x})-1)\frac{\partial a(\mathbf{x})}{\partial \theta}\right] + E_{\tilde{P}_0}\left[q(\mathbf{x})\frac{\partial a(\mathbf{x})}{\partial \theta}\right]\right) \\ &\approx \frac{1}{4}\left(-E_{\tilde{P}_1}\left[\frac{\partial a(\mathbf{x})}{\partial \theta}\right] + E_{\tilde{P}_0}\left[\frac{\partial a(\mathbf{x})}{\partial \theta}\right]\right) \end{aligned} \tag{5.31}$$

where the last equality is when the classifier is highly regularized: when the output weights are small, $a(\mathbf{x})$ is close to 0 and $q(\mathbf{x}) \approx 1/2$, so that $(1 - q(\mathbf{x})) \approx q(\mathbf{x})$. This expression for the log-likelihood gradient corresponds exactly to the one obtained for energy-based models where the likelihood is expressed in terms of a free energy (Equation (5.10)), when we interpret training examples from $\tilde{P}_1$ as positive examples ($y = 1$) (i.e., $\tilde{P}_1 = \hat{P}$) and model samples as negative examples ($y = 0$, i.e., $\tilde{P}_0 = P$). The gradient is also similar in structure to the Contrastive Divergence gradient estimator (Equation (5.27)). One way to interpret this result is that if we could improve a classifier that separated training samples from model samples, we could improve the log-likelihood of the model, by putting more probability mass on the side of training samples. Practically, this could be achieved with a classifier whose discriminant function was defined as the free energy of a generative model (up to a multiplicative factor), and assuming one could obtain samples (possibly approximate) from the model. A particular variant of this idea has been used to justify a boosting-like incremental algorithm for adding experts in products of experts [201].

6

Greedy Layer-Wise Training of Deep Architectures

6.1 Layer-Wise Training of Deep Belief Networks

A Deep Belief Network [73] with ℓ layers models the joint distribution between observed vector $\mathbf{x}$ and ℓ hidden layers $\mathbf{h}^k$ as follows:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = \left(\prod_{k=0}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell), \tag{6.1}$$

where $\mathbf{x} = \mathbf{h}^0$, $P(\mathbf{h}^{k-1}|\mathbf{h}^k)$ is a visible-given-hidden conditional distribution in an RBM associated with level k of the DBN, and $P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$ is the joint distribution in the top-level RBM. This is illustrated in Figure 6.1.

The conditional distributions $P(\mathbf{h}^k|\mathbf{h}^{k+1})$ and the top-level joint (an RBM) $P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$ define the generative model. In the following we introduce the letter Q for exact or approximate posteriors of that model, which are used for inference and training. The Q posteriors are all approximate except for the top level $Q(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$ which is equal to the true $P(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$ because $(\mathbf{h}^\ell, \mathbf{h}^{\ell-1})$ form an RBM, where exact inference is possible.

When we train the DBN in a greedy layer-wise fashion, as illustrated with the pseudo-code of Algorithm 2, each layer is initialized

Algorithm 2

`TrainUnsupervisedDBN`($\widehat{P}, \epsilon, \ell, W, \mathbf{b}, \mathbf{c}$, `mean_field_computation`)
Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM (e.g., by Contrastive Divergence).
$\widehat{P}$ is the input training distribution for the network
ϵ is a learning rate for the RBM training
ℓ is the number of layers to train
W^k is the weight matrix for level k, for k from 1 to ℓ
$\mathbf{b}^k$ is the visible units offset vector for RBM at level k, for k from 1 to ℓ
$\mathbf{c}^k$ is the hidden units offset vector for RBM at level k, for k from 1 to ℓ
`mean_field_computation` is a Boolean that is true iff training data at each additional level is obtained by a mean-field approximation instead of stochastic sampling

for $k = 1$ to ℓ **do**
  • initialize $W^k = 0$, $\mathbf{b}^k = 0$, $\mathbf{c}^k = 0$
  while not stopping criterion **do**
    • sample $\mathbf{h}^0 = \mathbf{x}$ from $\widehat{P}$
    for $i = 1$ to $k - 1$ **do**
      if `mean_field_computation then`
        • `assign` $\mathbf{h}^i_j$ `to` $Q(\mathbf{h}^i_j = 1|\mathbf{h}^{i-1})$`, for all elements` j `of` $\mathbf{h}^i$
      `else`
        • `sample` $\mathbf{h}^i_j$ `from` $Q(\mathbf{h}^i_j|\mathbf{h}^{i-1})$`, for all elements` j `of` $\mathbf{h}^i$
      `end if`
    end for
    • `RBMupdate`($\mathbf{h}^{k-1}, \epsilon, W^k, \mathbf{b}^k, \mathbf{c}^k$) {thus providing $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ for future use}
  end while
end for

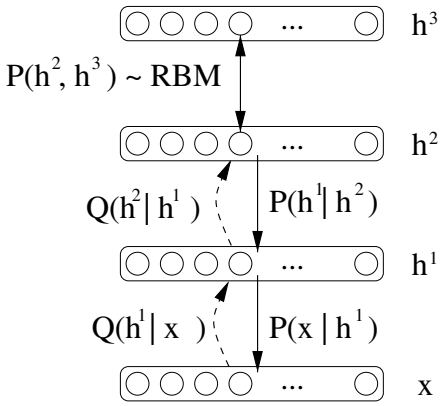

Fig. 6.1 Deep Belief Network as a generative model (generative path with P distributions, full arcs) and a means to extract multiple levels of representation of the input (recognition path with Q distributions, dashed arcs). The top two layers $\mathbf{h}^2$ and $\mathbf{h}^3$ form an RBM (for their joint distribution). The lower layers form a directed graphical model (sigmoid belief net $\mathbf{h}^2 \Rightarrow \mathbf{h}^1 \Rightarrow \mathbf{x}$) and the prior for the penultimate layer $\mathbf{h}^2$ is provided by the top-level RBM. $Q(\mathbf{h}^{k+1}|\mathbf{h}^k)$ approximates $P(\mathbf{h}^{k+1}|\mathbf{h}^k)$ but can be computed easily.

as an RBM, and we denote $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ the k-th RBM trained in this way, whereas $P(\ldots)$ denotes probabilities according to the DBN. We will use $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ as an approximation of $P(\mathbf{h}^k|\mathbf{h}^{k-1})$, because it is easy to compute and sample from $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ (which factorizes), and not from $P(\mathbf{h}^k|\mathbf{h}^{k-1})$ (which does not). These $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ can also be used to construct a representation of the input vector $\mathbf{x}$. To obtain an approximate posterior or representation for all the levels, we use the following procedure. First sample $\mathbf{h}^1 \sim Q(\mathbf{h}^1|\mathbf{x})$ from the first-level RBM, or alternatively with a mean-field approach use $\widehat{\mathbf{h}}^1 = E[\mathbf{h}^1|\mathbf{x}]$ instead of a sample of $\mathbf{h}^1$, where the expectation is over the RBM distribution $Q(\mathbf{h}^1|\mathbf{x})$. This is just the vector of output probabilities of the hidden units, in the common case where they are binomial units: $\widehat{\mathbf{h}}^1_i = \text{sigm}(\mathbf{b}^1 + W^1_i \mathbf{x})$. Taking either the mean-field vector $\widehat{\mathbf{h}}^1$ or the sample $\mathbf{h}^1$ as input for the second-level RBM, compute $\widehat{\mathbf{h}}^2$ or a sample $\mathbf{h}^2$, etc. until the last layer. Once a DBN is trained as per Algorithm 2, the parameters W^i (RBM weights) and $\mathbf{c}^i$ (RBM hidden unit offsets) for each layer can be used to initialize a deep multi-layer neural network. These parameters can then be fine-tuned with respect to another criterion (typically a supervised learning criterion).

A sample of the DBN generative model for $\mathbf{x}$ can be obtained as follows:

1. Sample a visible vector $\mathbf{h}^{\ell-1}$ from the top-level RBM. This can be achieved approximately by running a Gibbs chain in that RBM alternating between $\mathbf{h}^{\ell} \sim P(\mathbf{h}^{\ell}|\mathbf{h}^{\ell-1})$ and $\mathbf{h}^{\ell-1} \sim P(\mathbf{h}^{\ell-1}|\mathbf{h}^{\ell})$, as outlined in Section 5.3.1. By starting the chain from a representation $\mathbf{h}^{\ell-1}$ obtained from a training set example (through the $Q's$ as above), fewer Gibbs steps might be required.
2. For $k = \ell - 1$ down to 1, sample $\mathbf{h}^{k-1}$ given $\mathbf{h}^{k}$ according to the level-k hidden-to-visible conditional distribution $P(\mathbf{h}^{k-1}|\mathbf{h}^{k})$.
3. $\mathbf{x} = \mathbf{h}^{0}$ is the DBN sample.

6.2 Training Stacked Auto-Encoders

Auto-Encoders have been used as building blocks to build and initialize a deep multi-layer neural network [17, 99, 153, 195]. The training procedure is similar to the one for Deep Belief Networks:

1. Train the first layer as an auto-encoder to minimize some form of reconstruction error of the raw input. This is purely unsupervised.
2. The hidden units' outputs (i.e., the codes) of the auto-encoder are now used as input for another layer, also trained to be an auto-encoder. Again, we only need unlabeled examples.
3. Iterate as in step (2) to initialize the desired number of additional layers.
4. Take the last hidden layer output as input to a supervised layer and initialize its parameters (either randomly or by supervised training, keeping the rest of the network fixed).
5. Fine-tune all the parameters of this deep architecture with respect to the supervised criterion. Alternately, unfold all the auto-encoders into a very deep auto-encoder and fine-tune the global reconstruction error, as in [75].

The hope is that the unsupervised pre-training in this greedy layer-wise fashion has put the parameters of all the layers in a region of parameter space from which a good[1] local optimum can be reached by local descent. This indeed appears to happen in a number of tasks [17, 99, 153, 195].

The principle is exactly the same as the one previously proposed for training DBNs, but using auto-encoders instead of RBMs. Comparative experimental results suggest that Deep Belief Networks typically have an edge over Stacked Auto-Encoders [17, 99, 195]. This may be because CD-k is closer to the log-likelihood gradient than the reconstruction error gradient. However, since the reconstruction error gradient has less variance than CD-k (because no sampling is involved), it might be interesting to combine the two criteria, at least in the initial phases of learning. Note also that the DBN advantage disappeared in experiments where the ordinary auto-encoder was replaced by a denoising auto-encoder [195], which is stochastic (see Section 7.2).

An advantage of using auto-encoders instead of RBMs as the unsupervised building block of a deep architecture is that almost any parametrization of the layers is possible, as long as the training criterion is continuous in the parameters. On the other hand, the class of probabilistic models for which CD or other known tractable estimators of the log-likelihood gradient can be applied is currently more limited. A disadvantage of Stacked Auto-Encoders is that they do not correspond to a generative model: with generative models such as RBMs and DBNs, samples can be drawn to check qualitatively what has been learned, e.g., by visualizing the images or word sequences that the model sees as plausible.

6.3 Semi-Supervised and Partially Supervised Training

With DBNs and Stacked Auto-Encoders two kinds of training signals are available, and can be combined: the local layer-wise unsupervised training signal (from the RBM or auto-encoder associated with the layer), and a global supervised training signal (from the

[1] Good at least in the sense of generalization.

deep multi-layer network sharing the same parameters as the DBN or Stacked Auto-Encoder). In the algorithms presented above, the two training signals are used in sequence: first an unsupervised training phase, and second a supervised fine-tuning phase. Other combinations are possible.

One possibility is to combine both signals during training, and this is called partially supervised training in [17]. It has been found useful [17] when the true input distribution $P(X)$ is believed to be not strongly related to $P(Y|X)$. To make sure that an RBM preserves information relevant to Y in its hidden representation, the CD update is combined with the classification log-probability gradient, and for some distributions better predictions are thus obtained.

An appealing generalization of semi-supervised learning, especially in the context of deep architectures, is *self-taught learning* [109, 148], in which the unlabeled examples potentially come from classes other than the labeled classes. This is more realistic than the standard semi-supervised setting, e.g., even if we are only interested in some specific object classes, one can much more easily obtain unlabeled examples of arbitrary objects from the web (whereas it would be expensive to select only those pertaining to those selected classes of interest).

7

Variants of RBMs and Auto-Encoders

We review here some of the variations that have been proposed on the basic RBM and auto-encoder models to extend and improve them.

We have already mentioned that it is straightforward to generalize the conditional distributions associated with visible or hidden units in RBMs, e.g., to any member of the exponential family [200]. Gaussian units and exponential or truncated exponential units have been proposed or used in [17, 51, 99, 201]. With respect to the analysis presented here, the equations can be easily adapted by simply changing the domain of the sum (or integral) for the $\mathbf{h}_i$ and $\mathbf{x}_i$. Diagonal quadratic terms (e.g., to yield Gaussian or truncated Gaussian distributions) can also be added in the energy function without losing the property that the free energy factorizes.

7.1 Sparse Representations in Auto-Encoders and RBMs

Sparsity has become a concept of great interest recently, not only in machine learning but also in statistics and signal processing, in particular with the work on compressed sensing [30, 47], but it was introduced earlier in computational neuroscience in the context of sparse coding in

the visual system [139], and has been a key element deep convolutional networks exploiting of a variant of auto-encoders [121, 150, 151, 152, 153] with a sparse distributed representation, and has become a key ingredient in Deep Belief Networks [110].

7.1.1 Why a Sparse Representation?

We argue here that if one is going to have fixed-size representations, then sparse representations are more efficient (than non-sparse ones) in an information-theoretic sense, allowing for varying the effective number of bits per example. According to learning theory [117, 193], to obtain good generalization it is enough that the total number of bits needed to encode the *whole training set* be small, compared to the size of the training set. In many domains of interest different examples require different number of bits when compressed.

On the other hand, dimensionality reduction algorithms, whether linear such as PCA and ICA, or non-linear such as LLE and Isomap, map each example to the same low-dimensional space. In light of the above argument, it would be more efficient to map each example to a variable-length representation. To simplify the argument, assume this representation is a binary vector. If we are required to map each example to a fixed-length representation, a good solution would be to choose that representation to have enough degrees of freedom to represent the vast majority of the examples, while at the same allowing to compress that fixed-length bit vector to a smaller variable-size code for most of the examples. We now have two representations: the fixed-length one, which we might use as input to make predictions and make decisions, and a smaller, variable-size one, which can in principle be obtained from the fixed-length one through a compression step. For example, if the bits in our fixed-length representation vector have a high probability of being 0 (i.e., a sparsity condition), then for most examples it is easy to compress the fixed-length vector (in average by the amount of sparsity). For a given level of sparsity, the number of configurations of sparse vectors is much smaller than when less sparsity (or none at all) is imposed, so the entropy of sparser codes is smaller.

Another argument in favor of sparsity is that the fixed-length representation is going to be used as input for further processing, so that it should be easy to interpret. A highly compressed encoding is usually highly entangled, so that no subset of bits in the code can really be interpreted unless all the other bits are taken into account. Instead, we would like our fixed-length sparse representation to have the property that individual bits or small subsets of these bits can be interpreted, i.e., correspond to meaningful aspects of the input, and capture factors of variation in the data. For example, with a speech signal as input, if some bits encode the speaker characteristics and other bits encode generic features of the phoneme being pronounced, we have disentangled some of the factors of variation in the data, and some subset of the factors might be sufficient for some particular prediction tasks.

Another way to justify sparsity of the representation was proposed in [150], in the context of models based on auto-encoders. This view actually explains how one might get good models even though the partition function is not explicitly minimized, or only minimized approximately, as long as other constraints (such as sparsity) are used on the learned representation. Suppose that the representation learned by an auto-encoder is sparse, then the auto-encoder cannot reconstruct well every possible input pattern, because the number of sparse configurations is necessarily smaller than the number of dense configurations. To minimize the average reconstruction error on the training set, the auto-encoder then has to find a representation which captures statistical regularities of the data distribution. First of all, [150] connect the free energy with a form of reconstruction error (when one replaces summing over hidden unit configurations by maximizing over them). Minimizing reconstruction error on the training set therefore amounts to minimizing free energy, i.e., maximizing the numerator of an energy-based model likelihood (Equation (5.7)). Since the denominator (the partition function) is just a sum of the numerator over all possible input configurations, maximizing likelihood roughly amounts to making reconstruction error high for most possible input configurations, while making it low for those in the training set. This can be achieved if the encoder (which maps an input to its representation) is constrained in such a way that it cannot represent well most of the possible input

patterns (i.e., the reconstruction error *must be high* for most of the possible input configurations). Note how this is already achieved when the code is much smaller than the input. Another approach is to impose a sparsity penalty on the representation [150], which can be incorporated in the training criterion. In this way, the term of the log-likelihood gradient associated with the partition function is completely avoided, and replaced by a sparsity penalty on the hidden unit code. Interestingly, this idea could potentially be used to improve CD-k RBM training, which only uses an *approximate* estimator of the gradient of the log of the partition function. If we add a sparsity penalty to the hidden representation, we may compensate for the weaknesses of that approximation, by making sure we increase the free energy of most possible input configurations, and not only of the reconstructed neighbors of the input example that are obtained in the negative phase of Contrastive Divergence.

7.1.2 Sparse Auto-Encoders and Sparse Coding

There are many ways to enforce some form of sparsity on the hidden layer representation. The first successful deep architectures exploiting sparsity of representation involved auto-encoders [153]. Sparsity was achieved with a so-called sparsifying logistic, by which the codes are obtained with a nearly saturating logistic whose offset is adapted to maintain a low average number of times the code is significantly non-zero. One year later the same group introduced a somewhat simpler variant [150] based on a Student-t prior on the codes. The Student-t prior has been used in the past to obtain sparsity of the MAP estimates of the codes generating an input [139] in computational neuroscience models of the V1 visual cortex area. Another approach also connected to computational neuroscience involves two levels of sparse RBMs [110]. Sparsity is achieved with a regularization term that penalizes a deviation of the expected activation of the hidden units from a fixed low level. Whereas [139] had already shown that one level of sparse coding of images led to filters very similar to those seen in V1, [110] find that when training a sparse Deep Belief Network (i.e., two sparse RBMs on top of each other), the second level appears to learn to detect visual

features similar to those observed in area V2 of visual cortex (i.e., the area that follows area V1 in the main chain of processing of the visual cortex of primates).

In the compressed sensing literature sparsity is achieved with the ℓ_1 penalty on the codes, i.e., given bases in matrix W (each column of W is a basis) we typically look for codes $\mathbf{h}$ such that the input signal $\mathbf{x}$ is reconstructed with low ℓ_2 reconstruction error while $\mathbf{h}$ is sparse:

$$\min_h ||\mathbf{x} - W\mathbf{h}||_2^2 + \lambda||\mathbf{h}||_1, \tag{7.1}$$

where $||\mathbf{h}||_1 = \sum_i |\mathbf{h}_i|$. The actual number of non-zero components of $\mathbf{h}$ would be given by the ℓ_0 norm, but minimizing with it is combinatorially difficult, and the ℓ_1 norm is the closest p-norm that is also convex, making the overall minimization in Equation (7.1) convex. As is now well understood [30, 47], the ℓ_1 norm is a very good proxy for the ℓ_0 norm and naturally induces sparse results, and it can even be shown to *recover exactly* the true sparse code (if there is one), under mild conditions. Note that the ℓ_1 penalty corresponds to a Laplace prior, and that the posterior does not have a point mass at 0, but because of the above properties, the *mode* of the posterior (which is recovered when minimizing Equation (7.1)) is often at 0. Although minimizing Equation (7.1) is convex, minimizing jointly the codes and the decoder bases W is not convex, but has been done successfully with many different algorithms [46, 58, 116, 121, 139, 148].

Like directed graphical models (such as the sigmoid belief networks discussed in Section 4.4), sparse coding performs a kind of *explaining away*: it chooses one configuration (among many) of the hidden codes that could explain the input. These different configurations compete, and when one is selected, the others are completely turned off. This can be seen both as an advantage and as a disadvantage. The advantage is that if a cause is much more probable than the other, than it is the one that we want to highlight. The disadvantage is that it makes the resulting codes somewhat unstable, in the sense that small perturbations of the input $\mathbf{x}$ could give rise to very different values of the optimal code $\mathbf{h}$. This instability could spell trouble for higher levels of learned transformations or a trained classifier that would take $\mathbf{h}$ as input. Indeed it could make generalization more difficult if very similar inputs can end

up being represented very differently in the sparse code layer. There is also a computational weakness of these approaches that some authors have tried to address. Even though optimizing Equation (7.1) is efficient it can be hundreds of time slower than the kind of computation involved in computing the codes in ordinary auto-encoders or RBMs, making both training and recognition very slow. Another issue connected to the stability question is the joint optimization of the bases W with higher levels of a deep architecture. This is particularly important in view of the objective of fine-tuning the encoding so that it focuses on the most discriminant aspects of the signal. As discussed in Section 9.1.2, significant classification error improvements were obtained when fine-tuning all the levels of a deep architecture with respect to a discriminant criterion of interest. In principle one can compute gradients through the optimization of the codes, but if the result of the optimization is unstable, the gradient may not exist or be numerically unreliable. To address both the stability issue and the above fine-tuning issue, [6] propose to replace the ℓ_1 penalty by a softer approximation which only gives rise to approximately sparse coefficients (i.e., many very small coefficients, without actually converging to 0).

Keep in mind that sparse auto-encoders and sparse RBMs do not suffer from any of these sparse coding issues: computational complexity (of inferring the codes), stability of the inferred codes, and numerical stability and computational cost of computing gradients on the first layer in the context of global fine-tuning of a deep architecture. Sparse coding systems only parametrize the decoder: the encoder is defined implicitly as the solution of an optimization. Instead, an ordinary auto-encoder or an RBM has an encoder part (computing $P(\mathbf{h}|\mathbf{x})$) and a decoder part (computing $P(\mathbf{x}|\mathbf{h})$). A middle ground between ordinary auto-encoders and sparse coding is proposed in a series of papers on sparse auto-encoders [150, 151, 152, 153] applied in pattern recognition and machine vision tasks. They propose to let the codes $\mathbf{h}$ be free (as in sparse coding algorithms), but include a parametric encoder (as in ordinary auto-encoders and RBMs) and a penalty for the difference between the free non-parametric codes $\mathbf{h}$ and the outputs of the parametric encoder. In this way, the optimized codes $\mathbf{h}$ try to satisfy two objectives: reconstruct well the input (like in sparse coding), while not

being too far from the output of the encoder (which is stable by construction, because of the simple parametrization of the encoder). In the experiments performed, the encoder is just an affine transformation followed by a non-linearity like the sigmoid, and the decoder is linear as in sparse coding. Experiments show that the resulting codes work very well in the context of a deep architecture (with supervised fine-tuning) [150], and are more stable (e.g., with respect to slight perturbations of input images) than codes obtained by sparse coding [92].

7.2 Denoising Auto-Encoders

The denoising auto-encoder [195] is a stochastic version of the auto-encoder where the input is stochastically corrupted, but the uncorrupted input is still used as target for the reconstruction. Intuitively, a denoising auto-encoder does two things: try to encode the input (preserve the information about the input), and try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder. The latter can only be done by capturing the statistical dependencies between the inputs. In fact, in [195], the stochastic corruption process consists in randomly setting some of the inputs (as many as half of them) to zero. Hence the denoising auto-encoder is trying to predict the missing values from the non-missing values, for randomly selected subsets of missing patterns. The training criterion for denoising auto-encoders is expressed as a reconstruction log-likelihood,

$$-\log P(\mathbf{x}|\mathbf{c}(\tilde{\mathbf{x}})), \tag{7.2}$$

where $\mathbf{x}$ is the uncorrupted input, $\tilde{\mathbf{x}}$ is the stochastically corrupted input, and $\mathbf{c}(\tilde{\mathbf{x}})$ is the code obtained from $\tilde{\mathbf{x}}$. Hence the output of the decoder is viewed as the parameter for the above distribution (over the uncorrupted input). In the experiments performed [195], this distribution is factorized and binomial (one bit per pixel), and input pixel intensities are interpreted as probabilities. Note that a recurrent version of the denoising auto-encoder had been proposed earlier by [174], with corruption also corresponding to a form of occlusion (setting a rectangular region of the input image to 0). Using auto-encoders for denoising was actually introduced much earlier [103, 55]. The main innovation in [195] is therefore to show how this strategy is highly

successful as unsupervised pre-training for a deep architecture, and to link the denoising auto-encoder to a generative model.

Consider a random d-dimensional vector X, S a set of k indices, $X_S = (X_{S_1}, \ldots, X_{S_k})$ the sub-elements selected by S, and let X_{-S} all the sub-elements except those in S. Note that the set of conditional distributions $P(X_S|X_{-S})$ for some choices of S fully characterize the joint distribution $P(X)$, and this is exploited, for example, in Gibbs sampling. Note that bad things can happen when $|S| = 1$ and some pairs of input are perfectly correlated: the predictions can be perfect even though the joint has not really been captured, and this would correspond to a Gibbs chain that does not mix, i.e., does not converge. By considering random-size subsets and also insisting on reconstructing everything (like ordinary auto-encoders), this type of problem may be avoided in denoising auto-encoders.

Interestingly, in a series of experimental comparisons over 8 vision tasks, stacking denoising auto-encoders into a deep architecture fine-tuned with respect to a supervised criterion yielded generalization performance that was systematically better than stacking ordinary auto-encoders, and comparable or superior to Deep Belief Networks [195].

An interesting property of the denoising auto-encoder is that it can be shown to correspond to a generative model. Its training criterion is a bound on the log-likelihood of that generative model. Several possible generative models are discussed in [195]. A simple generative model is semi-parametric: sample a training example, corrupt it stochastically, apply the encoder function to obtain the hidden representation, apply the decoder function to it (obtaining parameters for a distribution over inputs), and sample an input. This is not very satisfying because it requires to keep the training set around (like non-parametric density models). Other possible generative models are explored in [195].

Another interesting property of the denoising auto-encoder is that it naturally lends itself to data with missing values or multi-modal data (when a subset of the modalities may be available for any particular example). This is because it is *trained* with inputs that have "missing" parts (when corruption consists in randomly hiding some of the input values).

7.3 Lateral Connections

The RBM can be made slightly less restricted by introducing interaction terms or "lateral connections" between visible units. Sampling $\mathbf{h}$ from $P(\mathbf{h}|\mathbf{x})$ is still easy but sampling $\mathbf{x}$ from $P(\mathbf{x}|\mathbf{h})$ is now generally more difficult, and amounts to sampling from a Markov Random Field which is also a fully observed Boltzmann machine, in which the offsets are dependent on the value of $\mathbf{h}$. [141] propose such a model for capturing image statistics and their results suggest that Deep Belief Nets (DBNs) based on such modules generate more realistic image patches than DBNs based on ordinary RBMs. Their results also show that the resulting distribution has marginal and pairwise statistics for pixel intensities that are similar to those observed on real image patches.

These lateral connections capture pairwise dependencies that can be more easily captured this way than using hidden units, saving the hidden units for capturing higher-order dependencies. In the case of the first layer, it can be seen that this amounts to a form of whitening, which has been found useful as a preprocessing step in image processing systems [139]. The idea proposed by [141] is to use lateral connections at all levels of a DBN (which can now be seen as a hierarchy of Markov random fields). The generic advantage of this type of approach would be that the higher level factors represented by the hidden units do not have to encode all the local "details" that the lateral connections at the levels below can capture. For example, when generating an image of a face, the approximate locations of the mouth and nose might be specified at a high level whereas their precise location could be selected in order to satisfy the pairwise preferences encoded in the lateral connections at a lower level. This appears to yield generated images with sharper edges and generally more accuracy in the relative locations of parts, without having to expand a large number of higher-level units.

In order to sample from $P(\mathbf{x}|\mathbf{h})$, we can start a Markov chain at the current example (which presumably already has pixel co-dependencies similar to those represented by the model, so that convergence should be quick) and only run a short chain on the $\mathbf{x}$'s (keeping $\mathbf{h}$ fixed). Denote U the square matrix of visible-to-visible connections, as per the general Boltzmann Machine energy function in Equation (5.15).

To reduce sampling variance in CD for this model, [141] used five damped mean-field steps instead of an ordinary Gibbs chain on the $\mathbf{x}$'s: $\mathbf{x}_t = \alpha \mathbf{x}_{t-1} + (1-\alpha)\text{sigm}(\mathbf{b} + U\mathbf{x}_{t-1} + W'\mathbf{h})$, with $\alpha \in (0,1)$.

7.4 Conditional RBMs and Temporal RBMs

A *Conditional RBM* is an RBM where some of the parameters are not free but are instead parametrized functions of a conditioning random variable. For example, consider an RBM for the joint distribution $P(\mathbf{x}, \mathbf{h})$ between observed vector $\mathbf{x}$ and hidden vector $\mathbf{h}$, with parameters $(\mathbf{b}, \mathbf{c}, W)$ as per Equation (5.15), respectively for input offsets $\mathbf{b}$, hidden units offsets $\mathbf{c}$, and the weight matrix W. This idea has been introduced by [182, 183] for context-dependent RBMs in which the hidden units offsets $\mathbf{c}$ are affine functions of a context variable $\mathbf{z}$. Hence the RBM represents $P(\mathbf{x}, \mathbf{h}|\mathbf{z})$ or, marginalizing over $\mathbf{h}$, $P(\mathbf{x}|\mathbf{z})$. In general the parameters $\theta = (\mathbf{b}, \mathbf{c}, W)$ of an RBM can be written as a parametrized function $\theta = f(\mathbf{z}; \omega)$, i.e., the actual free parameters of the conditional RBM with conditioning variable $\mathbf{z}$ are denoted ω. Generalizing RBMs to conditional RBMs allows building deep architectures in which the hidden variables at each level can be conditioned on the value of other variables (typically representing some form of context).

The Contrastive Divergence algorithm for RBMs can be easily generalized to the case of Conditional RBMs. The CD gradient estimator $\Delta\theta$ on a parameter θ can be simply back-propagated to obtain a gradient estimator on ω:

$$\Delta\omega = \Delta\theta \frac{\partial\theta}{\partial\omega}. \tag{7.3}$$

In the affine case $\mathbf{c} = \beta + M\mathbf{z}$ (with $\mathbf{c}$, β and $\mathbf{z}$ column vectors and M a matrix) studied by [183], the CD update on the conditional parameters is simply

$$\begin{aligned} \Delta\beta &= \Delta\mathbf{c}, \\ \Delta M &= \Delta\mathbf{c}\, \mathbf{z}', \end{aligned} \tag{7.4}$$

where the last multiplication is an outer product (applying the chain rule on derivatives), and $\Delta\mathbf{c}$ is the update given by CD-k on hidden units offsets.

This idea has been successfully applied to model conditional distributions $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \mathbf{x}_{t-3})$ in sequential data of human motion [183], where $\mathbf{x}_t$ is a vector of joint angles and other geometric features computed from motion capture data of human movements such as walking and running. Interestingly, this allows *generating* realistic human motion *sequences*, by successively sampling the t-th frame given the previously sampled k frames, i.e., approximating

$$P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) \approx P(\mathbf{x}_1, \ldots, \mathbf{x}_k) \prod_{t=k+1}^{T} P(\mathbf{x}_t|\mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k}). \quad (7.5)$$

The initial frames can be generated by using special null values as context or using a separate model for $P(\mathbf{x}_1, \ldots, \mathbf{x}_k)$.

As demonstrated by [126], it can be useful to make not just the offsets but also the weights conditional on a context variable. In that, case, we greatly increase the number of degrees of freedom, introducing the capability to model three-way interactions between an input unit $\mathbf{x}_i$, a hidden unit $\mathbf{h}_j$, and a context unit $\mathbf{z}_k$ through interaction parameters ζ_{ijk}. This approach has been used with $\mathbf{x}$ an image and $\mathbf{z}$ the previous image in a video, and the model learns to capture *flow fields* [126].

Probabilistic models of sequential data with hidden variables $\mathbf{h}_t$ (called *state*) can gain a lot by capturing the temporal dependencies between the hidden states at different times t in the sequence. This is what allows *Hidden Markov Models* (HMMs) [147] to capture dependencies in a long observed sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots$ even if the model only considers the hidden state sequence $\mathbf{h}_1, \mathbf{h}_2, \ldots$ to be a Markov chain of order 1 (where the direct dependence is only between $\mathbf{h}_t$ and $\mathbf{h}_{t+1}$). Whereas the hidden state representation $\mathbf{h}_t$ in HMMs is *local* (all the possible values of $\mathbf{h}_t$ are enumerated and specific parameters associated with each of these values), *Temporal RBMs* have been proposed [180] to construct a distributed representation of the state. The idea is an extension of the Conditional RBM presented above, but where the context includes not only past inputs but also past values of the state, e.g., we build a model of

$$P(\mathbf{h}_t, \mathbf{x}_t|\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, \ldots, \mathbf{h}_{t-k}, \mathbf{x}_{t-k}), \quad (7.6)$$

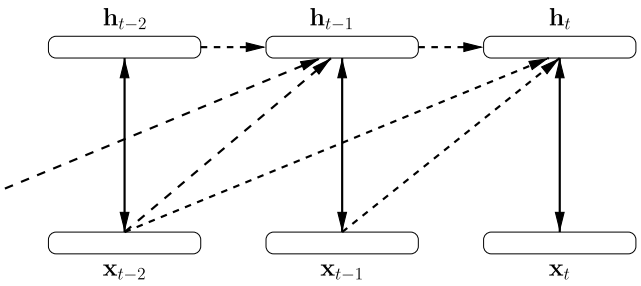

Fig. 7.1 Example of Temporal RBM for modeling sequential data, including dependencies between the hidden states. The double-arrow full arcs indicate an undirected connection, i.e., an RBM. The single-arrow dotted arcs indicate conditional dependency: the $(\mathbf{x}_t, \mathbf{h}_t)$ RBM is conditioned by the values of the past inputs and past hidden state vectors.

where the context is $\mathbf{z}_t = (\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, \ldots, \mathbf{h}_{t-k}, \mathbf{x}_{t-k})$, as illustrated in Figure 7.1. Although sampling of sequences generated by Temporal RBMs can be done as in Conditional RBMs (with the same MCMC approximation used to sample from RBMs, at each time step), exact inference of the hidden state sequence given an input sequence is no longer tractable. Instead, [180] propose to use a mean-field filtering approximation of the hidden sequence posterior.

7.5 Factored RBMs

In several probabilistic language models, it has been proposed to learn a distributed representation of each word [15, 16, 37, 43, 128, 130, 169, 170, 171, 207]. For an RBM that models a sequence of words, it would be convenient to have a parametrization that leads to automatically learning a distributed representation for each word in the vocabulary. This is essentially what [129] proposed. Consider an RBM input $\mathbf{x}$ that is the concatenation of one-hot vectors $\mathbf{v}_t$ for each word w_t in a fixed-size sequence $(w_1, w_2, \ldots, w_k)$, i.e., $\mathbf{v}_t$ contains all 0's except for a 1 at position w_t, and $\mathbf{x} = (\mathbf{v}_1', \mathbf{v}_2', \ldots, \mathbf{v}_k')'$. [129] use a factorization of the RBM weight matrix W into two factors, one that depends on the location t in the input subsequence, and one that does not. Consider the computation of the hidden units' probabilities given the input subsequence $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$. Instead of applying directly a matrix W to $\mathbf{x}$, do the following. First, each word symbol w_t is mapped through a matrix R

to a d-dimensional vector $R_{.,w_t} = R\mathbf{v}_t$, for $t \in \{1 \dots k\}$; second, the concatenated vectors $(R'_{.,w_1}, R'_{.,w_2}, \dots, R'_{.,w_k})'$ are multiplied by a matrix B. Hence $W = B\text{Diag}(R)$, where $\text{Diag}(R)$ is a block-diagonal matrix filled with R on the diagonal. This model has produced n-grams with better log-likelihood [129, 130], with further improvements in generalization performance when averaging predictions with state-of-the-art n-gram models [129].

7.6 Generalizing RBMs and Contrastive Divergence

Let us try to generalize the definition of RBM so as to include a large class of parametrizations for which essentially the same ideas and learning algorithms (such as Contrastive Divergence) that we have discussed above can be applied in a straightforward way. We generalize RBMs as follows: a *Generalized RBM* is an energy-based probabilistic model with input vector $\mathbf{x}$ and hidden vector $\mathbf{h}$ whose energy function is such that $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ both factorize. This definition can be formalized in terms of the parametrization of the energy function, which is also proposed by [73]:

Proposition 7.1. The energy function associated with a model of the form of Equation (5.5) such that $P(\mathbf{h}|\mathbf{x}) = \prod_i P(\mathbf{h}_i|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h}) = \prod_j P(\mathbf{x}_j|\mathbf{h})$ must have the form

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = \sum_j \phi_j(\mathbf{x}_j) + \sum_i \xi_i(\mathbf{h}_i) + \sum_{i,j} \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j). \tag{7.7}$$

This is a direct application of the Hammersley–Clifford theorem [33, 61]. [73] also showed that the above form is a necessary and sufficient condition to obtain *complementary priors*. Complementary priors allow the posterior distribution $P(\mathbf{h}|\mathbf{x})$ to factorize by a proper choice of $P(\mathbf{h})$.

In the case where the hidden and input values are binary, this new formulation does not actually bring any additional power of representation. Indeed, $\eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)$, which can take at most four different values according to the 2×2 configurations of $(\mathbf{h}_i, \mathbf{x}_j)$ could always be rewritten as a second order polynomial in $(\mathbf{h}_i, \mathbf{x}_j)$: $a + b\mathbf{x}_j + c\mathbf{h}_i + d\mathbf{h}_i\mathbf{x}_j$.

However, b and c can be folded into the offset terms and a into a global additive constant which does not matter (because it gets cancelled by the partition function).

On the other hand, when $\mathbf{x}$ or $\mathbf{h}$ are real vectors, one could imagine higher-capacity modeling of the $(\mathbf{h}_i, \mathbf{x}_j)$ interaction, possibly non-parametric, e.g., gradually adding terms to $\eta_{i,j}$ so as to better model the interaction. Furthermore, sampling from the conditional densities $P(\mathbf{x}_j|\mathbf{h})$ or $P(\mathbf{h}_i|\mathbf{x})$ would be tractable even if the $\eta_{i,j}$ are complicated functions, simply because these are one-dimensional densities from which efficient approximate sampling and numerical integration are easy to compute (e.g., by computing cumulative sums of the density over nested sub-intervals or bins).

This analysis also highlights the basic limitation of RBMs, which is that its parametrization only considers pairwise interactions between variables. It is because the $\mathbf{h}$ are hidden and because we can choose the number of hidden units, that we still have full expressive power over possible marginal distributions in $\mathbf{x}$ (in fact, we can represent any discrete distribution [102]). Other variants of RBMs discussed in Section 7.4 allow three-way interactions [126].

What would be a Contrastive Divergence update in this generalized RBM formulation? To simplify notations we note that the ϕ_j's and ξ_i's in Equation (7.7) can be incorporated within the $\eta_{i,j}$'s, so we ignore them in the following. Theorem 5.1 can still be applied with

$$\text{FreeEnergy}(\mathbf{x}) = -\log \sum_{\mathbf{h}} \exp\left(-\sum_{i,j} \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)\right).$$

The gradient of the free energy of a sample $\mathbf{x}$ is thus

$$\begin{aligned}
\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} &= \sum_{\mathbf{h}} \frac{\exp\left(-\sum_{i,j} \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)\right)}{\sum_{\tilde{\mathbf{h}}} \exp\left(-\sum_{i,j} \eta_{i,j}(\tilde{\mathbf{h}}_i, \mathbf{x}_j)\right)} \sum_{i,j} \frac{\partial \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)}{\partial \theta} \\
&= \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \sum_{i,j} \frac{\partial \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)}{\partial \theta} \\
&= E_{\mathbf{h}}\left[\sum_{i,j} \frac{\partial \eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j)}{\partial \theta} \middle| \mathbf{x}\right].
\end{aligned}$$

Thanks to Proposition 7.1, a Gibbs chain can still be run easily. Truncating the log-likelihood gradient expansion (Equation (5.28)) after k steps of the Gibbs chain, and approximating expectations with samples from this chain, one obtains an approximation of the log-likelihood gradient at training point $\mathbf{x}_1$ that depends only on Gibbs samples $\mathbf{h}_1$, $\mathbf{h}_{k+1}$ and $\mathbf{x}_{k+1}$:

$$\begin{aligned}\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} &\simeq -\frac{\partial \text{FreeEnergy}(\mathbf{x}_1)}{\partial \theta} + \frac{\partial \text{FreeEnergy}(\mathbf{x}_{k+1})}{\partial \theta}, \\ &\simeq \left(-\sum_{i,j} \frac{\partial \eta_{i,j}(\mathbf{h}_{1,i}, \mathbf{x}_{1,j})}{\partial \theta} + \sum_{i,j} \frac{\partial \eta_{i,j}(\mathbf{h}_{k+1,i}, \mathbf{x}_{k+1,j})}{\partial \theta} \right) \propto \Delta\theta,\end{aligned}$$

with $\Delta\theta$ the update rule for parameters θ of the model, corresponding to CD-k in such a generalized RBM. Note that in most parametrizations we would have a particular element of θ depend on $\eta_{i,j}$'s in such a way that no explicit sum is needed. For instance (taking expectation over $\mathbf{h}_{k+1}$ instead of sampling) we recover Algorithm 1 when

$$\eta_{i,j}(\mathbf{h}_i, \mathbf{x}_j) = -W_{ij}\mathbf{h}_i\mathbf{x}_j - \frac{\mathbf{b}_j\mathbf{x}_j}{n_h} - \frac{\mathbf{c}_i\mathbf{h}_i}{n_x},$$

where n_h and n_x are, respectively, the numbers of hidden and visible units, and we also recover the other variants described by [200, 17] for different forms of the energy and allowed set of values for hidden and input units.

8

Stochastic Variational Bounds for Joint Optimization of DBN Layers

In this section, we discuss mathematical underpinnings of algorithms for training a DBN as a whole. The log-likelihood of a DBN can be lower bounded using Jensen's inequality, and as we discuss below, this can justify the greedy layer-wise training strategy introduced in [73] and described in Section 6.1. We will use Equation (6.1) for a DBN joint distribution, writing $\mathbf{h}$ for $\mathbf{h}^1$ (the first level hidden vector) to lighten notation, and introducing an arbitrary conditional distribution $Q(\mathbf{h}|\mathbf{x})$. First multiply $\log P(\mathbf{x})$ by $1 = \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x})$, then use $P(\mathbf{x}) = P(\mathbf{x},\mathbf{h})/P(\mathbf{h}|\mathbf{x})$, and multiply by $1 = Q(\mathbf{h}|\mathbf{x})/Q(\mathbf{h}|\mathbf{x})$ and expand the terms:

$$\begin{aligned}
\log P(\mathbf{x}) &= \left(\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x})\right) \log P(\mathbf{x}) = \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log \frac{P(\mathbf{x},\mathbf{h})}{P(\mathbf{h}|\mathbf{x})} \\
&= \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log \frac{P(\mathbf{x},\mathbf{h})}{P(\mathbf{h}|\mathbf{x})} \frac{Q(\mathbf{h}|\mathbf{x})}{Q(\mathbf{h}|\mathbf{x})} \\
&= H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log P(\mathbf{x},\mathbf{h}) + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log \frac{Q(\mathbf{h}|\mathbf{x})}{P(\mathbf{h}|\mathbf{x})}
\end{aligned}$$

$$= KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x})) + H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h})\right), \quad (8.1)$$

where $H_{Q(\mathbf{h}|\mathbf{x})}$ is the entropy of the distribution $Q(\mathbf{h}|\mathbf{x})$. Non-negativity of the KL divergence gives the inequality

$$\log P(\mathbf{x}) \geq H_{Q(\mathbf{h}|\mathbf{x})} + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h})\right), \quad (8.2)$$

which becomes an equality when P and Q are identical, e.g., in the single-layer case (i.e., an RBM). Whereas we have chosen to use P to denote probabilities under the DBN, we use Q to denote probabilities under an RBM (the first level RBM), and in the equations choose $Q(\mathbf{h}|\mathbf{x})$ to be the hidden-given-visible conditional distribution of that first level RBM. We define that first level RBM such that $Q(\mathbf{x}|\mathbf{h}) = P(\mathbf{x}|\mathbf{h})$. In general $P(\mathbf{h}|\mathbf{x}) \neq Q(\mathbf{h}|\mathbf{x})$. This is because although the marginal $P(\mathbf{h})$ on the first layer hidden vector $\mathbf{h}^1 = \mathbf{h}$ is determined by the upper layers in the DBN, the RBM marginal $Q(\mathbf{h})$ only depends on the parameters of the RBM.

8.1 Unfolding RBMs into Infinite Directed Belief Networks

Before using the above decomposition of the likelihood to justify the greedy training procedure for DBNs, we need to establish a connection between $P(\mathbf{h}^1)$ in a DBN and the corresponding marginal $Q(\mathbf{h}^1)$ given by the first level RBM. The interesting observation is that there exists a DBN whose $\mathbf{h}^1$ marginal equals the first RBM's $\mathbf{h}^1$ marginal, i.e., $P(\mathbf{h}^1) = Q(\mathbf{h}^1)$, as long the dimension of $\mathbf{h}^2$ equals the dimension of $\mathbf{h}^0 = \mathbf{x}$. To see this, consider a second-level RBM whose weight matrix is the transpose of the first-level RBM (that is why we need the matching dimensions). Hence, by symmetry of the roles of visible and hidden in an RBM joint distribution (when transposing the weight matrix), the marginal distribution over the visible vector of the second RBM is equal to the marginal distribution $Q(\mathbf{h}^1)$ of the hidden vector of the first RBM.

Another interesting way to see this is given by [73]: consider the infinite Gibbs sampling Markov chain starting at $t = -\infty$ and terminating at $t = 0$, alternating between $\mathbf{x}$ and $\mathbf{h}^1$ for the first RBM, with visible vectors sampled on even t and hidden vectors on odd t. This chain can be seen as an infinite directed belief network with tied parameters (all even steps use weight matrix W' while all odd ones use weight matrix W). Alternatively, we can summarize any sub-chain from $t = -\infty$ to $t = \tau$ by an RBM with weight matrix W or W' according to the parity of τ, and obtain a DBN with $1 - \tau$ layers (not counting the input layer), as illustrated in Figure 8.1. This argument also shows that a two-layer DBN in which the second level has weights equal to the transpose of the first level weights is equivalent to a single RBM.

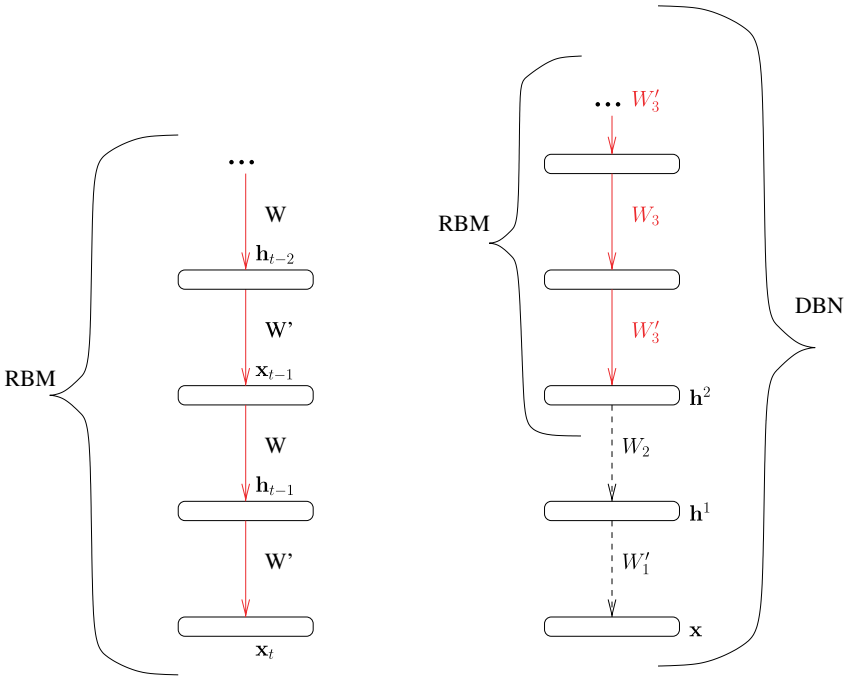

Fig. 8.1 An RBM can be unfolded as an infinite directed belief network with tied weights (see text). Left, the weight matrix W or its transpose are used depending on the parity of the layer index. This sequence of random variables corresponds to a Gibbs Markov chain to generate $\mathbf{x}_t$ (for t large). Right, the top-level RBM in a DBN can also be unfolded in the same way, showing that a DBN is an infinite directed graphical model in which *some* of the layers are tied (all except the bottom few ones).

8.2 Variational Justification of Greedy Layer-wise Training

Here, we discuss the argument made by [73] that adding one RBM layer improves the likelihood of a DBN. Let us suppose we have trained an RBM to model $\mathbf{x}$, which provides us with a model $Q(\mathbf{x})$ expressed through two conditionals $Q(\mathbf{h}^1|\mathbf{x})$ and $Q(\mathbf{x}|\mathbf{h}^1)$. Exploiting the argument in the previous subsection, let us now initialize an equivalent two-layer DBN, i.e., generating $P(\mathbf{x}) = Q(\mathbf{x})$, by taking $P(\mathbf{x}|\mathbf{h}^1) = Q(\mathbf{x}|\mathbf{h}^1)$ and $P(\mathbf{h}^1, \mathbf{h}^2)$ given by a second-level RBM whose weights are the transpose of the first-level RBM. Now let us come back to Equation (8.1) above, and the objective of improving the DBN likelihood by changing $P(\mathbf{h}^1)$, i.e., keeping $P(\mathbf{x}|\mathbf{h}^1)$ and $Q(\mathbf{h}^1|\mathbf{x})$ fixed but allowing the second level RBM to change. Interestingly, increasing the KL divergence term *increases* the likelihood. Starting from $P(\mathbf{h}^1|\mathbf{x}) = Q(\mathbf{h}^1|\mathbf{x})$, the KL term is zero (i.e., can only increase) and the entropy term in Equation (8.1) does not depend on the DBN $P(\mathbf{h}^1)$, so small improvements to the term with $P(\mathbf{h}^1)$ guarantee an increase in $\log P(\mathbf{x})$. We are also guaranteed that further improvements of the $P(\mathbf{h}^1)$ term (i.e., further training of the second RBM, detailed below) cannot bring the log-likelihood lower than it was before the second RBM was added. This is simply because of the positivity of the KL and entropy terms: further training of the second RBM increases a lower bound on the log-likelihood (Equation (8.2)), as argued by [73]. This justifies training the second RBM to maximize the second term, i.e., the expectation over the training set of $\sum_{\mathbf{h}^1} Q(\mathbf{h}^1|\mathbf{x}) \log P(\mathbf{h}^1)$.

The second-level RBM is thus trained to maximize

$$\sum_{\mathbf{x},\mathbf{h}^1} \hat{P}(\mathbf{x}) Q(\mathbf{h}^1|\mathbf{x}) \log P(\mathbf{h}^1), \tag{8.3}$$

with respect to $P(\mathbf{h}^1)$. This is the maximum-likelihood criterion for a model that sees examples $\mathbf{h}^1$ obtained as marginal samples from the joint distribution $\hat{P}(\mathbf{x})Q(\mathbf{h}^1|\mathbf{x})$. If we keep the first-level RBM fixed, then the second-level RBM could therefore be trained as follows: sample $\mathbf{x}$ from the training set, then sample $\mathbf{h}^1 \sim Q(\mathbf{h}^1|\mathbf{x})$, and consider that $\mathbf{h}^1$ as a training sample for the second-level RBM (i.e., as an observation for its 'visible' vector). If there was no constraint on $P(\mathbf{h}^1)$, the

maximizer of the above training criterion would be its "empirical" or target distribution

$$P^*(\mathbf{h}^1) = \sum_{\mathbf{x}} \hat{P}(\mathbf{x}) Q(\mathbf{h}^1|\mathbf{x}). \tag{8.4}$$

The same argument can be made to justify adding a third layer, etc. We obtain the greedy layer-wise training procedure outlined in Section 6.1. In practice the requirement that layer sizes alternate is not satisfied, and consequently neither is it common practice to initialize the newly added RBM with the transpose of the weights at the previous layer [73, 17], although it would be interesting to verify experimentally (in the case where the size constraint is imposed) whether the initialization with the transpose of the previous layer helps to speed up training.

Note that as we continue training the top part of the model (and this includes adding extra layers), there is no guarantee that $\log P(\mathbf{x})$ (in average over the training set) will monotonically increase. As our lower bound continues to increase, the actual log-likelihood could start decreasing. Let us examine more closely how this could happen. It would require the $KL(Q(\mathbf{h}^1|\mathbf{x})||P(\mathbf{h}^1|\mathbf{x}))$ term to decrease as the second RBM continues to be trained. However, this is unlikely in general: as the DBN's $P(\mathbf{h}^1)$ deviates more and more from the first RBM's marginal $Q(\mathbf{h}^1)$ on $\mathbf{h}^1$, it is likely that the posteriors $P(\mathbf{h}^1|\mathbf{x})$ (from the DBN) and $Q(\mathbf{h}^1|\mathbf{x})$ (from the RBM) deviate more and more (since $P(\mathbf{h}^1|\mathbf{x}) \propto Q(\mathbf{x}|\mathbf{h}^1)P(\mathbf{h}^1)$ and $Q(\mathbf{h}^1|\mathbf{x}) \propto Q(\mathbf{x}|\mathbf{h}^1)Q(\mathbf{h}^1)$), making the KL term in Equation (8.1) increase. As the training likelihood for the second RBM increases, $P(\mathbf{h}^1)$ moves smoothly from $Q(\mathbf{h}^1)$ towards $P^*(\mathbf{h}^1)$. Consequently, it seems very plausible that continued training of the second RBM is going to increase the DBN's likelihood (not just initially) and by transitivity, adding more layers will also likely increase the DBN's likelihood. However, it is not true that increasing the training likelihood for the second RBM starting from any parameter configuration guarantees that the DBN likelihood will increase, since at least one pathological counter-example can be found (I. Sutskever, personal communication). Consider the case where the first RBM has very large hidden biases, so that $Q(\mathbf{h}^1|\mathbf{x}) = Q(\mathbf{h}^1) = 1_{\mathbf{h}^1=\tilde{\mathbf{h}}} = P^*(\mathbf{h}^1)$, but large

weights and small visible offsets so that $P(\mathbf{x}_i|\mathbf{h}) = 1_{\mathbf{x}_i=\mathbf{h}_i}$, i.e., the hidden vector is copied to the visible units. When initializing the second RBM with the transpose of the weights of the first RBM, the training likelihood of the second RBM cannot be improved, nor can the DBN likelihood. However, if the second RBM was started from a "worse" configuration (worse in the sense of its training likelihood, and also worse in the sense of the DBN likelihood), then $P(\mathbf{h}^1)$ would move towards $P^*(\mathbf{h}^1) = Q(\mathbf{h}^1)$, making the second RBM likelihood improve while the KL term would decrease and the DBN likelihood would decrease. These conditions could not happen when initializing the second RBM properly (with a copy of the first RBM). So it remains an open question whether we can find conditions (excluding the above) which guarantee that while the likelihood of the second RBM increases, the DBN likelihood also increases.

Another argument to explain why the greedy procedure works is the following (Hinton, NIPS'2007 tutorial). The training distribution for the second RBM (samples $\mathbf{h}^1$ from $P^*(\mathbf{h}^1)$) looks more like data generated by an RBM than the original training distribution $\hat{P}(\mathbf{x})$. This is because $P^*(\mathbf{h}^1)$ was obtained by applying one sub-step of an RBM Gibbs chain on examples from $\hat{P}(\mathbf{x})$, and we know that applying many Gibbs steps would yield data from that RBM.

Unfortunately, when we train within this greedy layer-wise procedure an RBM that will not be the top-level level of a DBN, we are not taking into account the fact that more capacity will be added later to improve the prior on the hidden units. [102] have proposed considering alternatives to Contrastive Divergence for training RBMs destined to initialize intermediate layers of a DBN. The idea is to consider that $P(\mathbf{h})$ will be modeled with a very high capacity model (the higher levels of the DBN). In the limit case of infinite capacity, one can write down what that optimal $P(\mathbf{h})$ will be: it is simply the stochastic transformation of the empirical distribution through the stochastic mapping $Q(\mathbf{h}|\mathbf{x})$ of the first RBM (or previous RBMs), i.e., P^* of Equation (8.4) in the case of the second level. Plugging this back into the expression for $\log P(\mathbf{x})$, one finds that a good criterion for training the first RBM is the KL divergence between the data distribution and the distribution of the stochastic reconstruction vectors after one step of the Gibbs

chain. Experiments [102] confirm that this criterion yields better optimization of the DBN (initialized with this RBM). Unfortunately, this criterion is not tractable since it involves summing over all configurations of the hidden vector $\mathbf{h}$. Tractable approximations of it might be considered, since this criterion looks like a form of reconstruction error on a stochastic auto-encoder (with a generative model similar to one proposed for denoising auto-encoders [195]). Another interesting alternative, explored in the next section, is to directly work on joint optimization of all the layers of a DBN.

8.3 Joint Unsupervised Training of All the Layers

We discuss here how one could train a whole deep architecture such as a DBN in an unsupervised way, i.e., to represent well the input distribution.

8.3.1 The Wake–Sleep Algorithm

The Wake–Sleep algorithm [72] was introduced to train sigmoidal belief networks (i.e., where the distribution of the top layer units factorizes). It is based on a "recognition" model $Q(\mathbf{h}|\mathbf{x})$ (along with $Q(\mathbf{x})$ set to be the training set distribution) that acts as a variational approximation to the generative model $P(\mathbf{h}, \mathbf{x})$. Here, we denote with $\mathbf{h}$ all the hidden layers together. In a DBN, $Q(\mathbf{h}|\mathbf{x})$ is as defined above (Section 6.1), obtained by stochastically propagating samples upward (from input to higher layers) at each layer. In the Wake–Sleep algorithm, we decouple the recognition parameters (upward weights, used to compute $Q(\mathbf{h}|\mathbf{x})$) from the generative parameters (downward weights, used to compute $P(\mathbf{x}|\mathbf{h})$). The basic idea of the algorithm is simple:

1. **Wake phase**: sample $\mathbf{x}$ from the training set, generate $\mathbf{h} \sim Q(\mathbf{h}|\mathbf{x})$ and use this $(\mathbf{h}, \mathbf{x})$ as fully observed data for training $P(\mathbf{x}|\mathbf{h})$ and $P(\mathbf{h})$. This corresponds to doing one stochastic gradient step with respect to

$$\sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \log P(\mathbf{x}, \mathbf{h}). \tag{8.5}$$

2. **Sleep phase**: sample $(\mathbf{h}, \mathbf{x})$ from the model $P(\mathbf{x}, \mathbf{h})$, and use that pair as fully observed data for training $Q(\mathbf{h}|\mathbf{x})$. This corresponds to doing one stochastic gradient step with respect to

$$\sum_{\mathbf{h},\mathbf{x}} P(\mathbf{h}, \mathbf{x}) \log Q(\mathbf{h}|\mathbf{x}). \tag{8.6}$$

The Wake–Sleep algorithm has been used for DBNs in [73], after the weights associated with each layer have been trained as RBMs as discussed earlier. For a DBN with layers $(\mathbf{h}^1, \ldots, \mathbf{h}^\ell)$, the Wake phase updates for the weights of the top RBM (between $\mathbf{h}^{\ell-1}$ and $\mathbf{h}^\ell$) is done by considering the $\mathbf{h}^{\ell-1}$ sample (obtained from $Q(\mathbf{h}|\mathbf{x})$) as training data for the top RBM.

A variational approximation can be used to justify the Wake–Sleep algorithm. The log-likelihood decomposition in Equation (8.1)

$$\begin{aligned} \log P(\mathbf{x}) = {} & KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x})) + H_{Q(\mathbf{h}|\mathbf{x})} \\ & + \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h})\right), \end{aligned} \tag{8.7}$$

shows that the log-likelihood can be bounded from below by the opposite of the Helmholtz free energy [72, 53] F:

$$\log P(\mathbf{x}) = KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x})) - F(\mathbf{x}) \geq -F(\mathbf{x}), \tag{8.8}$$

where

$$F(\mathbf{x}) = -H_{Q(\mathbf{h}|\mathbf{x})} - \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{x}) \left(\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h})\right), \tag{8.9}$$

and the inequality is tight when $Q = P$. The variational approach is based on maximizing the lower bound $-F$ while trying to make the bound tight, i.e., *minimizing* $KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x}))$. When the bound is tight, an increase of $-F(\mathbf{x})$ is more likely to yield an increase of $\log P(\mathbf{x})$. Since we decouple the parameters of Q and of P, we can now see what the two phases are doing. In the Wake phase we consider Q fixed and do a stochastic gradient step towards maximizing the expected value of $F(\mathbf{x})$ over samples $\mathbf{x}$ of the training set, with respect

to parameters of P (i.e., we do not care about the entropy of Q). In the Sleep phase we would ideally like to make Q as close to P as possible in the sense of minimizing $KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x}))$ (i.e., taking Q as the reference), but instead we minimize $KL(P(\mathbf{h},\mathbf{x})||Q(\mathbf{h},\mathbf{x}))$, taking P as the reference, because $KL(Q(\mathbf{h}|\mathbf{x})||P(\mathbf{h}|\mathbf{x}))$ is intractable.

8.3.2 Transforming the DBN into a Boltzmann Machine

Another approach was recently proposed, yielding in the evaluated cases results superior to the use of the Wake–Sleep algorithm [161]. After initializing each layer as an RBM as already discussed in Section 6.1, the DBN is transformed into a corresponding deep Boltzmann machine. Because in a Boltzmann machine each unit receives input from above as well as from below, it is proposed to halve the RBM weights when initializing the deep Boltzmann machine from the layer-wise RBMs. It is very interesting to note that the RBM initialization of the deep Boltzmann machine was crucial to obtain the good results reported. The authors then propose approximations for the positive phase and negative phase gradients of the Boltzmann machine (see Section 5.2 and Equation (5.16)). For the positive phase (which in principle requires holding $\mathbf{x}$ fixed and sampling from $P(\mathbf{h}|\mathbf{x})$), they propose a variational approximation corresponding to a mean-field relaxation (propagating probabilities associated with each unit given the others, rather than samples, and iterating a few dozen times to let them settle). For the negative phase (which in principle requires sampling from the joint $P(\mathbf{h},\mathbf{x})$) they propose to use the idea of a persistent MCMC chain already discussed in Section 5.4.1 and introduced in [187]. The idea is to keep a set of $(\mathbf{h},\mathbf{x})$ states (or particles) that are updated by one Gibbs step according to the current model (i.e., sample each unit according to its probability given all the others at the previous step). Even though the parameters keep changing (very slowly), we continue the same Markov chain instead of starting a new one (as in the old Boltzmann machine algorithm [77, 1, 76]). This strategy seems to work very well, and [161] report an improvement over DBNs on the MNIST dataset, both in terms of data log-likelihood (estimated using annealed importance sampling [163]) and in terms of classification error

(after supervised fine-tuning), bringing down the error rate from 1.2% to 0.95%. More recently, [111] also transform the trained DBN into a deep Boltzmann machine in order to generate samples from it, and here the DBN has a convolutional structure.

9

Looking Forward

9.1 Global Optimization Strategies

As discussed Section 4.2, part of the explanation for the better generalization observed with layer-local unsupervised pre-training in deep architectures could well be that they help to better optimize the lower layers (near the input), by initializing supervised training in regions of parameter space associated with better unsupervised models. Similarly, initializing each layer of a deep Boltzmann machine as an RBM was important to achieve the good results reported [161]. In both settings, we optimize a proxy criterion that is layer-local before fine-tuning with respect to the whole deep architecture.

Here, we draw connections between existing work and approaches that could help to deal with difficult optimization problems, based on the principle of *continuation methods* [3]. Although they provide no guarantee to obtain the global optimum, these methods have been particularly useful in computational chemistry to find approximate solutions to difficult optimization problems involving the configurations of molecules [35, 132, 206]. The basic idea is to first solve an easier and smoothed version of the problem and gradually consider less smoothing, with the intuition that a smooth version of the problem reveals the

global picture, just like with simulated annealing [93]. One defines a single-parameter family of cost functions $C_\lambda(\theta)$ such that C_0 can be optimized more easily (maybe convex in θ), while C_1 is the criterion that we actually wish to minimize. One first minimizes $C_0(\theta)$ and then gradually increases λ while keeping θ at a local minimum of $C_\lambda(\theta)$. Typically C_0 is a highly smoothed version of C_1, so that θ gradually moves into the basin of attraction of a dominant (if not global) minimum of C_1.

9.1.1 Greedy Layer-wise Training of DBNs as a Continuation Method

The greedy layer-wise training algorithm for DBNs described in Section 6.1 can be viewed as an approximate continuation method, as follows. First of all recall (Section 8.1) that the top-level RBM of a DBN can be unfolded into an infinite directed graphical model with tied parameters. At each step of the greedy layer-wise procedure, we untie the parameters of the top-level RBM from the parameters of the penultimate level. So one can view the layer-wise procedure as follows. The model structure remains the same, an infinite chain of sigmoid belief layers, but we change the constraint on the parameters at each step of the layer-wise procedure. Initially all the layers are tied. After training the first RBM (i.e., optimizing under this constraint), we untie the first level parameters from the rest. After training the second RBM (i.e., optimizing under this slightly relaxed constraint), we untie the second level parameters from the rest, etc. Instead of a continuum of training criteria, we have a discrete sequence of (presumably) gradually more difficult optimization problems. By making the process greedy we fix the parameters of the first k levels after they have been trained and only optimize the $(k+1)$th, i.e., train an RBM. For this analogy to be strict we would need to initialize the weights of the newly added RBM with the transpose of the previous one. Note also that instead of optimizing all the parameters, the greedy layer-wise approach only optimizes the new ones. But even with these approximations, this analysis suggests an explanation for the good performance of the layer-wise training approach in terms of reaching better solutions.

9.1.2 Unsupervised to Supervised Transition

The experiments reported in many papers clearly show that an unsupervised pre-training followed by a supervised fine-tuning works very well for deep architectures. Whereas previous work on combining supervised and unsupervised criteria [100] focus on the regularization effect of an unsupervised criterion (and unlabeled examples, in semi-supervised learning), the discussion of Section 4.2 suggests that part of the gain observed with unsupervised pre-training of deep networks may arise out of better optimization of the lower layers of the deep architecture.

Much recent work has focused on starting from an unsupervised representation learning algorithm (such as sparse coding) and fine-tuning the representation with a discriminant criterion or combining the discriminant and unsupervised criteria [6, 97, 121].

In [97], an RBM is trained with a two-part visible vector that includes both the input $\mathbf{x}$ and the target class y. Such an RBM can either be trained to model the joint $P(\mathbf{x}, y)$ (e.g., by Contrastive Divergence) or to model the conditional $P(y|\mathbf{x})$ (the exact gradient of the conditional log-likelihood is tractable). The best results reported [97] combine both criteria, but the model is initialized using the non-discriminant criterion.

In [6, 121] the task of training the decoder bases in a sparse coding system is coupled with a task of training a classifier on to of the sparse codes. After initializing the decoder bases using non-discriminant learning, they can be fine-tuned using a discriminant criterion that is applied jointly on the representation parameters (i.e., the first layer bases, that gives rise to the sparse codes) and a set of classifier parameters (e.g., a linear classifier that takes the representation codes as input). According to [121], trying to directly optimize the supervised criterion without first initializing with non-discriminant training yielded very poor results. In fact, they propose a *smooth transition* from the non-discriminant criterion to the discriminant one, hence performing a kind of continuation method to optimize the discriminant criterion.

9.1.3 Controlling Temperature

Even optimizing the log-likelihood of a single RBM might be a difficult optimization problem. It turns out that the use of stochastic

gradient (such as the one obtained from CD-k) and small initial weights is again close to a continuation method, and could easily be turned into one. Consider the family of optimization problems corresponding to the *regularization path* [64] for an RBM, e.g., with ℓ_2 regularization of the parameters, the family of training criteria parametrized by $\lambda \in (0, 1]$:

$$C_\lambda(\theta) = -\sum_i \log P_\theta(\mathbf{x}_i) - ||\theta||^2 \log \lambda. \tag{9.1}$$

When $\lambda \to 0$, we have $\theta \to 0$, and it can be shown that the RBM log-likelihood becomes convex in θ. When $\lambda \to 1$, there is no regularization (note that some intermediate value of λ might be better in terms of generalization, if the training set is small). Controlling the magnitude of the offsets and weights in an RBM is equivalent to controlling the *temperature* in a Boltzmann machine (a scaling coefficient for the energy function). High temperature corresponds to a highly stochastic system, and at the limit a factorial and uniform distribution over the input. Low temperature corresponds to a more deterministic system where only a small subset of possible configurations are plausible.

Interestingly, one observes routinely that stochastic gradient descent starting from small weights gradually allows the weights to increase in magnitude, thus approximately following the regularization path. *Early stopping* is a well-known and efficient capacity control technique based on monitoring performance on a validation set during training and keeping the best parameters in terms of validation set error. The mathematical connection between early stopping and ℓ_2 regularization (along with margin) has already been established [36, 211]: starting from small parameters and doing gradient descent yields gradually larger parameters, corresponding to a gradually less regularized training criterion. However, with ordinary stochastic gradient descent (with no explicit regularization term), there is no guarantee that we would be tracking the sequence of local minima associated with a sequence of values of λ in Equation (9.1). It might be possible to slightly change the stochastic gradient algorithm to make it track better the regularization path, (i.e., make it closer to a continuation method), by controlling λ explicitly, gradually increasing λ when the optimization is near enough a local minimum for the current value of λ. Note that the same technique

might be extended for other difficult non-linear optimization problems found in machine learning, such as training a deep supervised neural network. We want to start from a globally optimal solution and gradually track local minima, starting from heavy regularization and moving slowly to little or none.

9.1.4 Shaping: Training with a Curriculum

Another continuation method may be obtained by gradually transforming the training task, from an easy one (maybe convex) where examples illustrate the simpler concepts, to the target one (with more difficult examples). Humans need about two decades to be trained as fully functional adults of our society. That training is highly organized, based on an education system and a curriculum which introduces different concepts at different times, exploiting previously learned concepts to ease the learning of new abstractions. The idea of training a learning machine with a curriculum can be traced back at least to [49]. The basic idea is to *start small*, learn easier aspects of the task or easier sub-tasks, and then gradually increase the difficulty level. From the point of view of building representations, advocated here, the idea is to learn representations that capture low-level abstractions first, and then exploit them and compose them to learn slightly higher-level abstractions necessary to explain more complex structure in the data. By choosing which examples to present and in which order to present them to the learning system, one can *guide* training and remarkably increase the speed at which learning can occur. This idea is routinely exploited in *animal training* and is called *shaping* [95, 144, 177].

Shaping and the use of a curriculum can also be seen as continuation methods. For this purpose, consider the learning problem of modeling the data coming from a training distribution $\hat{P}$. The idea is to reweigh the probability of sampling the examples from the training distribution, according to a given schedule that starts from the "easiest" examples and moves gradually towards examples illustrating more abstract concepts. At point t in the schedule, we train from distribution $\hat{P}_t$, with $\hat{P}_1 = \hat{P}$ and $\hat{P}_0$ chosen to be easy to learn. Like in any continuation

method, we move along the schedule when the learner has reached a local minimum at the current point t in the schedule, i.e., when it has sufficiently mastered the previously presented examples (sampled from $\hat{P}_t$). Making small changes in t corresponds to smooth changes in the probability of sampling examples in the training distribution, so we can construct a continuous path starting from an easy learning problem and ending in the desired training distribution. This idea is developed further in [20], with experiments showing better generalization obtained when training with a curriculum leading to a target distribution, compared to training only with the target distribution, on both vision and language tasks.

There is a connection between the shaping/curriculum idea and the greedy layer-wise idea. In both cases we want to exploit the notion that a high level abstraction can more conveniently be learned once appropriate lower-level abstractions have been learned. In the case of the layer-wise approach, this is achieved by gradually adding more capacity in a way that builds upon previously learned concepts. In the case of the curriculum, we control the training examples so as to make sure that the simpler concepts have actually been learned before showing many examples of the more advanced concepts. Showing complicated illustrations of the more advanced concepts is likely to be generally a waste of time, as suggested by the difficulty for humans to grasp a new idea if they do not first understand the concepts necessary to express that new idea compactly.

With the curriculum idea we introduce a teacher, in addition to the learner and the training distribution or environment. The teacher can use two sources of information to decide on the schedule: (a) prior knowledge about a sequence of concepts that can more easily be learned when presented in that order, and (b) monitoring of the learner's progress to decide when to move on to new material from the curriculum. The teacher has to select a level of difficulty for new examples which is a compromise between "too easy" (the learner will not need to change its model to account for these examples) and "too hard" (the learner cannot make an incremental change that can account for these examples so they will most likely be treated as outliers or special cases, i.e., not helping generalization).

9.2 Why Unsupervised Learning is Important

One of the claims of this monograph is that powerful unsupervised or semi-supervised (or self-taught) learning is a crucial component in building successful learning algorithms for deep architectures aimed at approaching AI. We briefly cover the arguments in favor of this hypothesis here:

- Scarcity of labeled examples and availability of many unlabeled examples (possibly not only of the classes of interest, as in self-taught learning [148]).
- Unknown future tasks: if a learning agent does not know what future learning tasks it will have to deal with in the future, but it knows that the task will be defined with respect to a world (i.e., random variables) that it can observe now, it would appear very rational to collect and integrate as much information as possible about this world so as to learn what makes it tick.
- Once a good high-level representation is learned, other learning tasks (e.g., supervised or reinforcement learning) could be much easier. We know for example that kernel machines can be very powerful if using an appropriate kernel, i.e., an appropriate feature space. Similarly, we know powerful reinforcement learning algorithms which have guarantees in the case where the actions are essentially obtained through linear combination of appropriate features. We do not know what the appropriate representation should be, but one would be reassured if it captured the salient factors of variation in the input data, and disentangled them.
- Layer-wise unsupervised learning: this was argued in Section 4.3. Much of the learning could be done using information available locally in one layer or sub-layer of the architecture, thus avoiding the hypothesized problems with supervised gradients propagating through long chains with large fan-in elements.
- Connected to the two previous points is the idea that unsupervised learning could put the parameters of a supervised

or reinforcement learning machine in a region from which gradient descent (local optimization) would yield good solutions. This has been verified empirically in several settings, in particular in the experiment of Figure 4.2 and in [17, 98, 50].

- The extra constraints imposed on the optimization by requiring the model to capture not only the input-to-target dependency but also the statistical regularities of the input distribution might be helpful in avoiding some poorly generalizing apparent local minima (those that do not correspond to good modeling of the input distribution). Note that in general extra constraints may also create more local minima, but we observe experimentally [17] that both training and test error can be reduced by unsupervised pre-training, suggesting that the unsupervised pre-training moves the parameters in a region of space closer to local minima corresponding to learning better representations (in the lower layers). It has been argued [71] (but is debatable) that unsupervised learning is less prone to overfitting than supervised learning. Deep architectures have typically been used to construct a supervised classifier, and in that case the unsupervised learning component can clearly be seen as a regularizer or a prior [137, 100, 118, 50] that forces the resulting parameters to make sense not only to model classes given inputs but also to capture the structure of the input distribution.

9.3 Open Questions

Research on deep architectures is still young and many questions remain unanswered. The following are potentially interesting.

1. Can the results pertaining to the role of computational depth in circuits be generalized beyond logic gates and linear threshold units?

2. Is there a depth that is mostly sufficient for the computations necessary to approach human-level performance of AI tasks?
3. How can the theoretical results on depth of circuits with a fixed size input be generalized to dynamical circuits operating in time, with context and the possibility of recursive computation?
4. Why is gradient-based training of deep neural networks from random initialization often unsuccessful?
5. Are RBMs trained by CD doing a good job of preserving the information in their input (since they are not trained as auto-encoders they might lose information about the input that may turn out to be important later), and if not how can that be fixed?
6. Is the supervised training criterion for deep architectures (and maybe the log-likelihood in deep Boltzmann machines and DBNs) really fraught with actual poor local minima or is it just that the criterion is too intricate for the optimization algorithms tried (such as gradient descent and conjugate gradients)?
7. Is the presence of local minima an important issue in training RBMs?
8. Could we replace RBMs and auto-encoders by algorithms that would be proficient at extracting good representations but involving an easier optimization problem, perhaps even a convex one?
9. Current training algorithms for deep architectures involves many phases (one per layer, plus a global fine-tuning). This is not very practical in the purely online setting since once we have moved into fine-tuning, we might be trapped in an apparent local minimum. Is it possible to come up with a completely online procedure for training deep architectures that preserves an unsupervised component all along? Note that [202] is appealing for this reason.
10. Should the number of Gibbs steps in Contrastive Divergence be adjusted during training?

11. Can we significantly improve upon Contrastive Divergence, taking computation time into account? New alternatives have recently been proposed which deserve further investigation [187, 188].
12. Besides reconstruction error, are there other more appropriate ways to monitor progress during training of RBMs and DBNs? Equivalently, are there tractable approximations of the partition function in RBMs and DBNs? Recent work in this direction [163, 133] using annealed importance sampling is encouraging.
13. Could RBMs and auto-encoders be improved by imposing some form of sparsity penalty on the representations they learn, and what are the best ways to do so?
14. Without increasing the number of hidden units, can the capacity of an RBM be increased using non-parametric forms of its energy function?
15. Since we only have a generative model for single denoising auto-encoders, is there a probabilistic interpretation to models learned in *Stacked* Auto-Encoders or *Stacked* Denoising Auto-Encoders?
16. How efficient is the greedy layer-wise algorithm for training Deep Belief Networks (in terms of maximizing the training data likelihood)? Is it too greedy?
17. Can we obtain low variance and low bias estimators of the log-likelihood gradient in Deep Belief Networks and related deep generative models, i.e., can we jointly train all the layers (with respect to the unsupervised objective)?
18. Unsupervised layer-level training procedures discussed here help training deep architectures, but experiments suggest that training still gets stuck in apparent local minima and cannot exploit all the information in very large datasets. Is it true? Can we go beyond these limitations by developing more powerful optimization strategies for deep architectures?

19. Can optimization strategies based on continuation methods deliver significantly improved training of deep architectures?
20. Are there other efficiently trainable deep architectures besides Deep Belief Networks, Stacked Auto-Encoders, and deep Boltzmann machines?
21. Is a curriculum needed to learn the kinds of high-level abstractions that humans take years or decades to learn?
22. Can the principles discovered to train deep architectures be applied or generalized to train recurrent networks or dynamical belief networks, which learn to represent context and long-term dependencies?
23. How can deep architectures be generalized to represent information that, by its nature, might seem not easily representable by vectors, because of its variable size and structure (e.g., trees, graphs)?
24. Although Deep Belief Networks are in principle well suited for the semi-supervised and self-taught learning settings, what are the best ways to adapt the current deep learning algorithms to these setting and how would they fare compared to existing semi-supervised algorithms?
25. When labeled examples are available, how should supervised and unsupervised criteria be combined to learn the model's representations of the input?
26. Can we find analogs of the computations necessary for Contrastive Divergence and Deep Belief Net learning in the brain?
27. The cortex is not at all like a feedforward neural network in that there are significant feedback connections (e.g., going back from later stages of visual processing to earlier ones) and these may serve a role not only in learning (as in RBMs) but also in integrating contextual priors with visual evidence [112]. What kind of models can give rise to such interactions in deep architectures, and learn properly with such interactions?

10

Conclusion

This monograph started with a number of motivations: first to use learning to approach AI, then on the intuitive plausibility of decomposing a problem into multiple levels of computation and representation, followed by theoretical results showing that a computational architecture that does not have enough of these levels can require a huge number of computational elements, and the observation that a learning algorithm that relies only on local generalization is unlikely to generalize well when trying to learn highly varying functions.

Turning to architectures and algorithms, we first motivated distributed representations of the data, in which a huge number of possible configurations of abstract features of the input are possible, allowing a system to compactly represent each example, while opening the door to a rich form of generalization. The discussion then focused on the difficulty of successfully training deep architectures for learning multiple levels of distributed representations. Although the reasons for the failure of standard gradient-based methods in this case remain to be clarified, several algorithms have been introduced in recent years that demonstrate much better performance than was previously possible with simple gradient-based optimization,

and we have tried to focus on the underlying principles behind their success.

Although much of this monograph has focused on deep neural net and deep graphical model architectures, the idea of exploring learning algorithms for deep architectures should be explored beyond the neural net framework. For example, it would be interesting to consider extensions of decision tree and boosting algorithms to multiple levels.

Kernel-learning algorithms suggest another path which should be explored, since a feature space that captures the abstractions relevant to the distribution of interest would be just the right space in which to apply the kernel machinery. Research in this direction should consider ways in which the learned kernel would have the ability to generalize non-locally, to avoid the curse of dimensionality issues raised in Section 3.1 when trying to learn a highly varying function.

The monograph focused on a particular family of algorithms, the Deep Belief Networks, and their component elements, the Restricted Boltzmann Machine, and very near neighbors: different kinds of auto-encoders, which can also be stacked successfully to form a deep architecture. We studied and connected together estimators of the log-likelihood gradient in Restricted Boltzmann machines, helping to justify the use of the Contrastive Divergence update for training Restricted Boltzmann Machines. We highlighted an optimization principle that has worked well for Deep Belief Networks and related algorithms such as Stacked Auto-Encoders, based on a greedy, layer-wise, unsupervised initialization of each level of the model. We found that this optimization principle is actually an approximation of a more general optimization principle, exploited in so-called continuation methods, in which a series of gradually more difficult optimization problems are solved. This suggested new avenues for optimizing deep architectures, either by tracking solutions along a regularization path, or by presenting the system with a sequence of selected examples illustrating gradually more complicated concepts, in a way analogous to the way students or animals are trained.

Acknowledgments

The author is particularly grateful for the inspiration from and constructive input from Yann LeCun, Aaron Courville, Olivier Delalleau, Dumitru Erhan, Pascal Vincent, Geoffrey Hinton, Joseph Turian, Hugo Larochelle, Nicolas Le Roux, Jérôme Louradour, Pascal Lamblin, James Bergstra, Pierre-Antoine Manzagol and Xavier Glorot. This research was performed thanks to the funding from NSERC, MITACS, and the Canada Research Chairs.

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive Science*, vol. 9, pp. 147–169, 1985.

[2] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. P. Xing, "Training hierarchical feed-forward visual recognition models using transfer learning from pseudo tasks," in *Proceedings of the 10th European Conference on Computer Vision (ECCV'08)*, pp. 69–82, 2008.

[3] E. L. Allgower and K. Georg, *Numerical Continuation Methods. An Introduction. No. 13 in Springer Series in Computational Mathematics*, Springer-Verlag, 1980.

[4] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan, "An introduction to MCMC for machine learning," *Machine Learning*, vol. 50, pp. 5–43, 2003.

[5] D. Attwell and S. B. Laughlin, "An energy budget for signaling in the grey matter of the brain," *Journal of Cerebral Blood Flow And Metabolism*, vol. 21, pp. 1133–1145, 2001.

[6] J. A. Bagnell and D. M. Bradley, "Differentiable sparse coding," in *Advances in Neural Information Processing Systems 21 (NIPS'08)*, (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), NIPS Foundation, 2009.

[7] J. Baxter, "Learning internal representations," in *Proceedings of the 8th International Conference on Computational Learning Theory (COLT'95)*, pp. 311–320, Santa Cruz, California: ACM Press, 1995.

[8] J. Baxter, "A Bayesian/information theoretic model of learning via multiple task sampling," *Machine Learning*, vol. 28, pp. 7–40, 1997.

[9] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *Proceedings of the 17th International Confer-*

ence on Computational Learning Theory (COLT'04), (J. Shawe-Taylor and Y. Singer, eds.), pp. 624–638, Springer, 2004.

[10] M. Belkin and P. Niyogi, "Using manifold structure for partially labeled classification," in *Advances in Neural Information Processing Systems 15 (NIPS'02)*, (S. Becker, S. Thrun, and K. Obermayer, eds.), Cambridge, MA: MIT Press, 2003.

[11] A. J. Bell and T. J. Sejnowski, "An information maximisation approach to blind separation and blind deconvolution," *Neural Computation*, vol. 7, no. 6, pp. 1129–1159, 1995.

[12] Y. Bengio and O. Delalleau, "Justifying and generalizing contrastive divergence," *Neural Computation*, vol. 21, no. 6, pp. 1601–1621, 2009.

[13] Y. Bengio, O. Delalleau, and N. Le Roux, "The Curse of highly variable functions for local kernel machines," in *Advances in Neural Information Processing Systems 18 (NIPS'05)*, (Y. Weiss, B. Schölkopf, and J. Platt, eds.), pp. 107–114, Cambridge, MA: MIT Press, 2006.

[14] Y. Bengio, O. Delalleau, and C. Simard, "Decision trees do not generalize to new variations," *Computational Intelligence*, To appear, 2009.

[15] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems 13 (NIPS'00)*, (T. Leen, T. Dietterich, and V. Tresp, eds.), pp. 933–938, MIT Press, 2001.

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[17] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 153–160, MIT Press, 2007.

[18] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte, "Convex neural networks," in *Advances in Neural Information Processing Systems 18 (NIPS'05)*, (Y. Weiss, B. Schölkopf, and J. Platt, eds.), pp. 123–130, Cambridge, MA: MIT Press, 2006.

[19] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," in *Large Scale Kernel Machines*, (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), MIT Press, 2007.

[20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the Twenty-sixth InternationalConference onMachine Learning (ICML09)*, (L. Bottou and M. Littman, eds.), pp. 41–48, Montreal: ACM, 2009.

[21] Y. Bengio, M. Monperrus, and H. Larochelle, "Non-local estimation of manifold structure," *Neural Computation*, vol. 18, no. 10, pp. 2509–2528, 2006.

[22] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[23] J. Bergstra and Y. Bengio, "Slow, decorrelated features for pretraining complex cell-like networks," in *Advances in Neural Information Processing Systems 22 (NIPS'09)*, (D. Schuurmans, Y. Bengio, C. Williams, J. Lafferty, and A. Culotta, eds.), December 2010.

[24] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, Pittsburgh: ACM, 1992.

[25] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, vol. 59, pp. 291–294, 1988.

[26] M. Brand, "Charting a manifold," in *Advances in Neural Information Processing Systems 15 (NIPS'02)*, (S. Becker, S. Thrun, and K. Obermayer, eds.), pp. 961–968, MIT Press, 2003.

[27] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.

[29] L. D. Brown, *Fundamentals of Statistical Exponential Families*. 1986. Vol. 9, Inst. of Math. Statist. Lecture Notes Monograph Series.

[30] E. Candes and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 15, no. 12, pp. 4203–4215, 2005.

[31] M. A. Carreira-Perpiñan and G. E. Hinton, "On contrastive divergence learning," in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, (R. G. Cowell and Z. Ghahramani, eds.), pp. 33–40, Society for Artificial Intelligence and Statistics, 2005.

[32] R. Caruana, "Multitask connectionist learning," in *Proceedings of the 1993 Connectionist Models Summer School*, pp. 372–379, 1993.

[33] P. Clifford, "Markov random fields in statistics," in *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*, (G. Grimmett and D. Welsh, eds.), pp. 19–32, Oxford University Press, 1990.

[34] D. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," in *Advances in Neural Information Processing Systems 7 (NIPS'94)*, (G. Tesauro, D. Touretzky, and T. Leen, eds.), pp. 705–712, Cambridge MA: MIT Press, 1995.

[35] T. F. Coleman and Z. Wu, "Parallel continuation-based global optimization for molecular conformation and protein folding," Technical Report Cornell University, Dept. of Computer Science, 1994.

[36] R. Collobert and S. Bengio, "Links between perceptrons, MLPs and SVMs," in *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*, (C. E. Brodley, ed.), p. 23, New York, NY, USA: ACM, 2004.

[37] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 160–167, ACM, 2008.

[38] C. Cortes, P. Haffner, and M. Mohri, "Rational kernels: Theory and algorithms," *Journal of Machine Learning Research*, vol. 5, pp. 1035–1062, 2004.

[39] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[40] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola, "On kernel-target alignment," in *Advances in Neural Information Processing Systems 14 (NIPS'01)*, (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), pp. 367–373, 2002.

[41] F. Cucker and D. Grigoriev, "Complexity lower bounds for approximation algebraic computation trees," *Journal of Complexity*, vol. 15, no. 4, pp. 499–512, 1999.

[42] P. Dayan, G. E. Hinton, R. Neal, and R. Zemel, "The Helmholtz machine," *Neural Computation*, vol. 7, pp. 889–904, 1995.

[43] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[44] O. Delalleau, Y. Bengio, and N. L. Roux, "Efficient non-parametric function induction in semi-supervised learning," in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, (R. G. Cowell and Z. Ghahramani, eds.), pp. 96–103, Society for Artificial Intelligence and Statistics, January 2005.

[45] G. Desjardins and Y. Bengio, "Empirical evaluation of convolutional RBMs for vision," Technical Report 1327, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2008.

[46] E. Doi, D. C. Balcan, and M. S. Lewicki, "A theoretical analysis of robust coding over noisy overcomplete channels," in *Advances in Neural Information Processing Systems 18 (NIPS'05)*, (Y. Weiss, B. Schölkopf, and J. Platt, eds.), pp. 307–314, Cambridge, MA: MIT Press, 2006.

[47] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[48] S. Duane, A. Kennedy, B. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Phys. Lett. B*, vol. 195, pp. 216–222, 1987.

[49] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, pp. 781–799, 1993.

[50] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, pp. 153–160, 2009.

[51] Y. Freund and D. Haussler, "Unsupervised learning of distributions on binary vectors using two layer networks," Technical Report UCSC-CRL-94-25, University of California, Santa Cruz, 1994.

[52] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of Thirteenth International Conference*, pp. 148–156, USA: ACM, 1996.

[53] B. J. Frey, G. E. Hinton, and P. Dayan, "Does the wake-sleep algorithm learn good density estimators?," in *Advances in Neural Information Processing Systems 8 (NIPS'95)*, (D. Touretzky, M. Mozer, and M. Hasselmo, eds.), pp. 661–670, Cambridge, MA: MIT Press, 1996.

[54] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[55] P. Gallinari, Y. LeCun, S. Thiria, and F. Fogelman-Soulie, "Memoires associatives distribuees," in *Proceedings of COGNITIVA 87*, Paris, La Villette, 1987.

[56] T. Gärtner, "A survey of kernels for structured data," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, pp. 49–58, 2003.

[57] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, November 1984.

[58] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng, "Shift-invariant sparse coding for audio classification," in *Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence (UAI'07)*, 2007.

[59] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'06)*, pp. 1735–1742, IEEE Press, 2006.

[60] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun, "Deep belief net learning in a long-range vision system for autonomous off-road driving," in *Proc. Intelligent Robots and Systems (IROS'08)*, pp. 628–633, 2008.

[61] J. M. Hammersley and P. Clifford, "Markov field on finite graphs and lattices," Unpublished manuscript, 1971.

[62] J. Håstad, "Almost optimal lower bounds for small depth circuits," in *Proceedings of the 18th annual ACM Symposium on Theory of Computing*, pp. 6–20, Berkeley, California: ACM Press, 1986.

[63] J. Håstad and M. Goldmann, "On the power of small-depth threshold circuits," *Computational Complexity*, vol. 1, pp. 113–129, 1991.

[64] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The entire regularization path for the support vector machine," *Journal of Machine Learning Research*, vol. 5, pp. 1391–1415, 2004.

[65] K. A. Heller and Z. Ghahramani, "A nonparametric bayesian approach to modeling overlapping clusters," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, pp. 187–194, San Juan, Porto Rico: Omnipress, 2007.

[66] K. A. Heller, S. Williamson, and Z. Ghahramani, "Statistical models for partial membership," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 392–399, ACM, 2008.

[67] G. Hinton and J. Anderson, *Parallel Models of Associative Memory*. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1981.

[68] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1–12, Amherst: Lawrence Erlbaum, Hillsdale, 1986.

[69] G. E. Hinton, "Products of experts," in *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN),* vol. 1, pp. 1–6, Edinburgh, Scotland: IEE, 1999.

[70] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, 2002.

[71] G. E. Hinton, "To recognize shapes, first learn to generate images," Technical Report UTML TR 2006-003, University of Toronto, 2006.
[72] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The wake-sleep algorithm for unsupervised neural networks," *Science*, vol. 268, pp. 1558–1161, 1995.
[73] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.
[74] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[75] G. E. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, 2006.
[76] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, (D. E. Rumelhart and J. L. McClelland, eds.), pp. 282–317, Cambridge, MA: MIT Press, 1986.
[77] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, 1984.
[78] G. E. Hinton, M. Welling, Y. W. Teh, and S. Osindero, "A new view of ICA," in *Proceedings of 3rd International Conference on Independent Component Analysis and Blind Signal Separation (ICA'01)*, pp. 746–751, San Diego, CA, 2001.
[79] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," in *Advances in Neural Information Processing Systems 6 (NIPS'93)*, (D. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 3–10, Morgan Kaufmann Publishers, Inc., 1994.
[80] T. K. Ho, "Random decision forest," in *3rd International Conference on Document Analysis and Recognition (ICDAR'95)*, pp. 278–282, Montreal, Canada, 1995.
[81] S. Hochreiter Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.
[82] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, pp. 417–441, 498–520, 1933.
[83] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex," *Journal of Physiology (London)*, vol. 160, pp. 106–154, 1962.
[84] A. Hyvärinen, "Estimation of non-normalized statistical models using score matching," *Journal of Machine Learning Research*, vol. 6, pp. 695–709, 2005.
[85] A. Hyvärinen, "Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables," *IEEE Transactions on Neural Networks*, vol. 18, pp. 1529–1531, 2007.
[86] A. Hyvärinen, "Some extensions of score matching," *Computational Statistics and Data Analysis*, vol. 51, pp. 2499–2512, 2007.

[87] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. Wiley-Interscience, May 2001.

[88] N. Intrator and S. Edelman, "How to make a low-dimensional representation suitable for diverse tasks," *Connection Science, Special issue on Transfer in Neural Networks*, vol. 8, pp. 205–224, 1996.

[89] T. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," Available from http://www.cse.ucsc.edu/ haussler/pubs.html, Preprint, Dept.of Computer Science, Univ. of California. A shorter version is in Advances in Neural Information Processing Systems 11, 1998.

[90] N. Japkowicz, S. J. Hanson, and M. A. Gluck, "Nonlinear autoassociation is not equivalent to PCA," *Neural Computation*, vol. 12, no. 3, pp. 531–545, 2000.

[91] M. I. Jordan, *Learning in Graphical Models*. Dordrecht, Netherlands: Kluwer, 1998.

[92] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "Fast inference in sparse coding algorithms with applications to object recognition," Technical Report, Computational and Biological Learning Lab, Courant Institute, NYU, Technical Report CBLL-TR-2008-12-01, 2008.

[93] S. Kirkpatrick, C. D. G. Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[94] U. Köster and A. Hyvärinen, "A two-layer ICA-like model estimated by score matching," in *Int. Conf. Artificial Neural Networks (ICANN'2007)*, pp. 798–807, 2007.

[95] K. A. Krueger and P. Dayan, "Flexible shaping: How learning in small steps helps," *Cognition*, vol. 110, pp. 380–394, 2009.

[96] G. Lanckriet, N. Cristianini, P. Bartlett, L. El Gahoui, and M. Jordan, "Learning the kernel matrix with semi-definite programming," in *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, (C. Sammut and A. G. Hoffmann, eds.), pp. 323–330, Morgan Kaufmann, 2002.

[97] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 536–543, ACM, 2008.

[98] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[99] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, (Z. Ghahramani, ed.), pp. 473–480, ACM, 2007.

[100] J. A. Lasserre, C. M. Bishop, and T. P. Minka, "Principled hybrids of generative and discriminative models," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'06)*, pp. 87–94, Washington, DC, USA, 2006. IEEE Computer Society.

[101] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[102] N. Le Roux and Y. Bengio, "Representational power of restricted boltzmann machines and deep belief networks," *Neural Computation*, vol. 20, no. 6, pp. 1631–1649, 2008.

[103] Y. LeCun, "Modèles connexionistes de l'apprentissage," PhD thesis, Université de Paris VI, 1987.

[104] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[105] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, (G. B. Orr and K.-R. Müller, eds.), pp. 9–50, Springer, 1998.

[106] Y. LeCun, S. Chopra, R. M. Hadsell, M.-A. Ranzato, and F.-J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, pp. 191–246, G. Bakir and T. Hofman and B. Scholkopf and A. Smola and B. Taskar: MIT Press, 2006.

[107] Y. LeCun and F. Huang, "Loss functions for discriminative training of energy-based models," in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, (R. G. Cowell and Z. Ghahramani, eds.), 2005.

[108] Y. LeCun, F.-J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'04),* vol. 2, pp. 97–104, Los Alamitos, CA, USA: IEEE Computer Society, 2004.

[109] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 801–808, MIT Press, 2007.

[110] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area V2," in *Advances in Neural Information Processing Systems 20 (NIPS'07)*, (J. Platt, D. Koller, Y. Singer, and S. P. Roweis, eds.), Cambridge, MA: MIT Press, 2008.

[111] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, (L. Bottou and M. Littman, eds.), Montreal (Qc), Canada: ACM, 2009.

[112] T.-S. Lee and D. Mumford, "Hierarchical bayesian inference in the visual cortex," *Journal of Optical Society of America, A*, vol. 20, no. 7, pp. 1434–1448, 2003.

[113] P. Lennie, "The cost of cortical computation," *Current Biology*, vol. 13, pp. 493–497, Mar 18 2003.

[114] I. Levner, *Data Driven Object Segmentation*. 2008. PhD thesis, Department of Computer Science, University of Alberta.

[115] M. Lewicki and T. Sejnowski, "Learning nonlinear overcomplete representations for efficient coding," in *Advances in Neural Information Processing Systems 10 (NIPS'97)*, (M. Jordan, M. Kearns, and S. Solla, eds.), pp. 556–562, Cambridge, MA, USA: MIT Press, 1998.

[116] M. S. Lewicki and T. J. Sejnowski, "Learning overcomplete representations," *Neural Computation*, vol. 12, no. 2, pp. 337–365, 2000.

[117] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*. New York, NY: Springer, Second ed., 1997.

[118] P. Liang and M. I. Jordan, "An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 584–591, New York, NY, USA: ACM, 2008.

[119] T. Lin, B. G. Horne, P. Tino, and C. L. Giles, "Learning long-term dependencies is not as difficult with NARX recurrent neural networks," Technical Report UMICAS-TR-95-78, Institute for Advanced Computer Studies, University of Mariland, 1995.

[120] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," in *Large Scale Kernel Machines*, (L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds.), pp. 301–320, Cambridge, MA: MIT Press, 2007.

[121] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning," in *Advances in Neural Information Processing Systems 21 (NIPS'08)*, (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 1033–1040, 2009. NIPS Foundation.

[122] J. L. McClelland and D. E. Rumelhart, "An interactive activation model of context effects in letter perception," *Psychological Review*, pp. 375–407, 1981.

[123] J. L. McClelland and D. E. Rumelhart, *Explorations in parallel distributed processing*. Cambridge: MIT Press, 1988.

[124] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* vol. 2. Cambridge: MIT Press, 1986.

[125] W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[126] R. Memisevic and G. E. Hinton, "Unsupervised learning of image transformations," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*, 2007.

[127] E. Mendelson, *Introduction to Mathematical Logic, 4th ed.* 1997. Chapman & Hall.

[128] R. Miikkulainen and M. G. Dyer, "Natural language processing with modular PDP networks and distributed lexicon," *Cognitive Science*, vol. 15, pp. 343–399, 1991.

[129] A. Mnih and G. E. Hinton, "Three new graphical models for statistical language modelling," in *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, (Z. Ghahramani, ed.), pp. 641–648, ACM, 2007.

[130] A. Mnih and G. E. Hinton, "A scalable hierarchical distributed language model," in *Advances in Neural Information Processing Systems 21 (NIPS'08)*, (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 1081–1088, 2009.

[131] H. Mobahi, R. Collobert, and J. Weston, "Deep Learning from temporal coherence in video," in *Proceedings of the 26th International Conference on Machine Learning*, (L. Bottou and M. Littman, eds.), pp. 737–744, Montreal: Omnipress, June 2009.

[132] J. More and Z. Wu, "Smoothing techniques for macromolecular global optimization," in *Nonlinear Optimization and Applications*, (G. D. Pillo and F. Giannessi, eds.), Plenum Press, 1996.

[133] I. Murray and R. Salakhutdinov, "Evaluating probabilities under high-dimensional latent variable models," in *Advances in Neural Information Processing Systems 21 (NIPS'08),* vol. 21, (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 1137–1144, 2009.

[134] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *International Journal of Computer Vision*, vol. 80, no. 1, pp. 45–57, 2008.

[135] R. M. Neal, "Connectionist learning of belief networks," *Artificial Intelligence*, vol. 56, pp. 71–113, 1992.

[136] R. M. Neal, "Bayesian learning for neural networks," PhD thesis, Department of Computer Science, University of Toronto, 1994.

[137] A. Y. Ng and M. I. Jordan, "On Discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in Neural Information Processing Systems 14 (NIPS'01)*, (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), pp. 841–848, 2002.

[138] J. Niebles and L. Fei-Fei, "A hierarchical model of shape and appearance for human action classification," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*, 2007.

[139] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: a strategy employed by V1?," *Vision Research*, vol. 37, pp. 3311–3325, December 1997.

[140] P. Orponen, "Computational complexity of neural networks: a survey," *Nordic Journal of Computing*, vol. 1, no. 1, pp. 94–110, 1994.

[141] S. Osindero and G. E. Hinton, "Modeling image patches with a directed hierarchy of Markov random field," in *Advances in Neural Information Processing Systems 20 (NIPS'07)*, (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 1121–1128, Cambridge, MA: MIT Press, 2008.

[142] B. Pearlmutter and L. C. Parra, "A context-sensitive generalization of ICA," in *International Conference On Neural Information Processing*, (L. Xu, ed.), pp. 151–157, Hong-Kong, 1996.

[143] E. Pérez and L. A. Rendell, "Learning despite concept variation by finding structure in attribute-based data," in *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, (L. Saitta, ed.), pp. 391–399, Morgan Kaufmann, 1996.

[144] G. B. Peterson, "A day of great illumination: B. F. Skinner's discovery of shaping," *Journal of the Experimental Analysis of Behavior*, vol. 82, no. 3, pp. 317–328, 2004.

[145] N. Pinto, J. DiCarlo, and D. Cox, "Establishing good benchmarks and baselines for face recognition," in *ECCV 2008 Faces in 'Real-Life' Images Workshop*, 2008. Marseille France, Erik Learned-Miller and Andras Ferencz and Frédéric Jurie.

[146] J. B. Pollack, "Recursive distributed representations," *Artificial Intelligence*, vol. 46, no. 1, pp. 77–105, 1990.

[147] L. R. Rabiner and B. H. Juang, "An Introduction to hidden Markov models," *IEEE ASSP Magazine*, pp. 257–285, january 1986.

[148] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, (Z. Ghahramani, ed.), pp. 759–766, ACM, 2007.

[149] M. Ranzato, Y. Boureau, S. Chopra, and Y. LeCun, "A unified energy-based framework for unsupervised learning," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, San Juan, Porto Rico: Omnipress, 2007.

[150] M. Ranzato, Y.-L. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," in *Advances in Neural Information Processing Systems 20 (NIPS'07)*, (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 1185–1192, Cambridge, MA: MIT Press, 2008.

[151] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*, IEEE Press, 2007.

[152] M. Ranzato and Y. LeCun, "A sparse and locally shift invariant feature extractor applied to document images," in *International Conference on Document Analysis and Recognition (ICDAR'07)*, pp. 1213–1217, Washington, DC, USA: IEEE Computer Society, 2007.

[153] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 1137–1144, MIT Press, 2007.

[154] M. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08),* vol. 307, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 792–799, ACM, 2008.

[155] S. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[156] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[157] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1, Cambridge: MIT Press, 1986.

[158] R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, San Juan, Porto Rico: Omnipress, 2007.

[159] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," in *Proceedings of the 2007 Workshop on Information Retrieval and applications of Graphical Models (SIGIR 2007)*, Amsterdam: Elsevier, 2007.

[160] R. Salakhutdinov and G. E. Hinton, "Using deep belief nets to learn covariance kernels for Gaussian processes," in *Advances in Neural Information Processing Systems 20 (NIPS'07)*, (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 1249–1256, Cambridge, MA: MIT Press, 2008.

[161] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann machines," in *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, vol. 5, pp. 448–455, 2009.

[162] R. Salakhutdinov, A. Mnih, and G. E. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, (Z. Ghahramani, ed.), pp. 791–798, New York, NY, USA: ACM, 2007.

[163] R. Salakhutdinov and I. Murray, "On the quantitative analysis of deep belief networks," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 872–879, ACM, 2008.

[164] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *Journal of Artificial Intelligence Research*, vol. 4, pp. 61–76, 1996.

[165] M. Schmitt, "Descartes' rule of signs for radial basis function neural networks," *Neural Computation*, vol. 14, no. 12, pp. 2997–3011, 2002.

[166] B. Schölkopf, C. J. C. Burges, and A. J. Smola, *Advances in Kernel Methods — Support Vector Learning*. Cambridge, MA: MIT Press, 1999.

[167] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola, "Input space versus feature space in kernel-based methods," *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[168] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, pp. 1299–1319, 1998.

[169] H. Schwenk, "Efficient training of large neural networks for language modeling," in *International Joint Conference on Neural Networks (IJCNN)*, pp. 3050–3064, 2004.

[170] H. Schwenk and J.-L. Gauvain, "Connectionist language modeling for large vocabulary continuous speech recognition," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 765–768, Orlando, Florida, 2002.

[171] H. Schwenk and J.-L. Gauvain, "Building continuous space language models for transcribing European languages," in *Interspeech*, pp. 737–740, 2005.

[172] H. Schwenk and M. Milgram, "Transformation invariant autoassociation with application to handwritten character recognition," in *Advances in Neural*

Information Processing Systems 7 (NIPS'94), (G. Tesauro, D. Touretzky, and T. Leen, eds.), pp. 991–998, MIT Press, 1995.

[173] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, "A quantitative theory of immediate visual recognition," *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, vol. 165, pp. 33–56, 2007.

[174] S. H. Seung, "Learning continuous attractors in recurrent networks," in *Advances in Neural Information Processing Systems 10 (NIPS'97)*, (M. Jordan, M. Kearns, and S. Solla, eds.), pp. 654–660, MIT Press, 1998.

[175] D. Simard, P. Y. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks," in *International Conference on Document Analysis and Recognition (ICDAR'03)*, p. 958, Washington, DC, USA: IEEE Computer Society, 2003.

[176] P. Y. Simard, Y. LeCun, and J. Denker, "Efficient pattern recognition using a new transformation distance," in *Advances in Neural Information Processing Systems 5 (NIPS'92)*, (C. Giles, S. Hanson, and J. Cowan, eds.), pp. 50–58, Morgan Kaufmann, San Mateo, 1993.

[177] B. F. Skinner, "Reinforcement today," *American Psychologist*, vol. 13, pp. 94–99, 1958.

[178] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing*, vol. 1, (D. E. Rumelhart and J. L. McClelland, eds.), pp. 194–281, Cambridge: MIT Press, 1986. ch. 6.

[179] E. B. Sudderth, A. Torralba, W. T. Freeman, and A. S. Willsky, "Describing visual scenes using transformed objects and parts," *International Journal of Computer Vision*, vol. 77, pp. 291–330, 2007.

[180] I. Sutskever and G. E. Hinton, "Learning multilevel distributed representations for high-dimensional sequences," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, San Juan, Porto Rico: Omnipress, 2007.

[181] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[182] G. Taylor and G. Hinton, "Factored conditional restricted Boltzmann machines for modeling motion style," in *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, (L. Bottou and M. Littman, eds.), pp. 1025–1032, Montreal: Omnipress, June 2009.

[183] G. Taylor, G. E. Hinton, and S. Roweis, "Modeling human motion using binary latent variables," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*, (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 1345–1352, Cambridge, MA: MIT Press, 2007.

[184] Y. Teh, M. Welling, S. Osindero, and G. E. Hinton, "Energy-based models for sparse overcomplete representations," *Journal of Machine Learning Research*, vol. 4, pp. 1235–1260, 2003.

[185] J. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[186] S. Thrun, "Is learning the n-th thing any easier than learning the first?," in *Advances in Neural Information Processing Systems 8 (NIPS'95)*, (D. Touretzky, M. Mozer, and M. Hasselmo, eds.), pp. 640–646, Cambridge, MA: MIT Press, 1996.

[187] T. Tieleman, "Training restricted Boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 1064–1071, ACM, 2008.

[188] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, (L. Bottou and M. Littman, eds.), pp. 1033–1040, New York, NY, USA: ACM, 2009.

[189] I. Titov and J. Henderson, "Constituent parsing with incremental sigmoid belief networks," in *Proc. 45th Meeting of Association for Computational Linguistics (ACL'07)*, pp. 632–639, Prague, Czech Republic, 2007.

[190] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large databases for recognition," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'08)*, pp. 1–8, 2008.

[191] P. E. Utgoff and D. J. Stracuzzi, "Many-layered learning," *Neural Computation*, vol. 14, pp. 2497–2539, 2002.

[192] L. van der Maaten and G. E. Hinton, "Visualizing data using t-Sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, November 2008.

[193] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[194] R. Vilalta, G. Blix, and L. Rendell, "Global data analysis and the fragmentation problem in decision tree induction," in *Proceedings of the 9th European Conference on Machine Learning (ECML'97)*, pp. 312–327, Springer-Verlag, 1997.

[195] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 1096–1103, ACM, 2008.

[196] L. Wang and K. L. Chan, "Learning kernel parameters by using class separability measure," 6th kernel machines workshop, in conjunction with Neural Information Processing Systems (NIPS), 2002.

[197] M. Weber, M. Welling, and P. Perona, "Unsupervised learning of models for recognition," in *Proc. 6th Europ. Conf. Comp. Vis., ECCV2000*, pp. 18–32, Dublin, 2000.

[198] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

[199] Y. Weiss, "Segmentation using eigenvectors: a unifying view," in *Proceedings IEEE International Conference on Computer Vision (ICCV'99)*, pp. 975–982, 1999.

[200] M. Welling, M. Rosen-Zvi, and G. E. Hinton, "Exponential family harmoniums with an application to information retrieval," in *Advances in Neural Information Processing Systems 17 (NIPS'04)*, (L. Saul, Y. Weiss, and L. Bottou, eds.), pp. 1481–1488, Cambridge, MA: MIT Press, 2005.

[201] M. Welling, R. Zemel, and G. E. Hinton, "Self-supervised boosting," in *Advances in Neural Information Processing Systems 15 (NIPS'02)*, (S. Becker, S. Thrun, and K. Obermayer, eds.), pp. 665–672, MIT Press, 2003.

[202] J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, (W. W. Cohen, A. McCallum, and S. T. Roweis, eds.), pp. 1168–1175, New York, NY, USA: ACM, 2008.

[203] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in neural information processing systems 8 (NIPS'95)*, (D. Touretzky, M. Mozer, and M. Hasselmo, eds.), pp. 514–520, Cambridge, MA: MIT Press, 1996.

[204] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.

[205] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–249, 1992.

[206] Z. Wu, "Global continuation for distance geometry problems," *SIAM Journal of Optimization*, vol. 7, pp. 814–836, 1997.

[207] P. Xu, A. Emami, and F. Jelinek, "Training connectionist models for the structured language model," in *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP'2003),* vol. 10, pp. 160–167, 2003.

[208] A. Yao, "Separating the polynomial-time hierarchy by oracles," in *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pp. 1–10, 1985.

[209] D. Zhou, O. Bousquet, T. Navin Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems 16 (NIPS'03)*, (S. Thrun, L. Saul, and B. Schölkopf, eds.), pp. 321–328, Cambridge, MA: MIT Press, 2004.

[210] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proceedings of the Twenty International Conference on Machine Learning (ICML'03)*, (T. Fawcett and N. Mishra, eds.), pp. 912–919, AAAI Press, 2003.

[211] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the Twenty International Conference on Machine Learning (ICML'03)*, (T. Fawcett and N. Mishra, eds.), pp. 928–936, AAAI Press, 2003.