Learning Deep Architectures for AI

Yoshua Bengio
Dept. IRO, Université de Montréal
C.P. 6128, Montreal, Qc, H3C 3J7, Canada
Yoshua.Bengio@umontreal.ca
http://www.iro.umontreal.ca/∼bengioy

Technical Report 1312

Abstract

Theoretical results strongly 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 needs *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 optimization 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 paper 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. The situation is similar for other AI tasks.

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. Consider for example the task of interpreting an input image such as the one in Figure 1. When humans try to solve a particular task in AI (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. 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. 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 object 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. In this case, our 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 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. 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 a particular velocity). 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.

We do not know exactly how to build robust **MAN** detectors or even intermediate abstractions that would be appropriate. Furthermore, the number of visual and semantic categories (such as **MAN**) that we would like an "intelligent" machine to capture is 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.

One of the important points we argue in the first part of this paper is that the functions learned should have a structure composed of multiple levels, analogous to the multiple levels of abstraction that humans naturally envision when they describe an aspect of their world. The arguments rest both on intuition and on theoretical results about the representational limitations of functions defined with an insufficient number of levels. Since most current work in machine learning is based on shallow architectures, these results suggest investigating learning algorithms for deep architectures, which is the subject of the second part of this paper.

In much of machine vision systems, learning algorithms have been limited to specific parts of such a processing chain. The rest of of design remains labor-intensive, which might limit the scale of such systems. On the other hand, a hallmark of what we would consider intelligent includes a large enough vocabulary 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. It would seem foolish not to exploit the underlying commonalities between these these tasks and between the concepts they require. This has been the focus of research on *multi-task learning* (Caruana, 1993; Baxter, 1995; Intrator & Edelman, 1996; Baxter, 1997). 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. 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 configuration of abstract features, generalization to unseen categories could follow naturally from new configurations of these features. Even though only some configurations of these features would be 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. This idea underlies the concept of *distributed representation* that is at the core of many of the learning algorithms described in this paper, and discussed in Section 4.

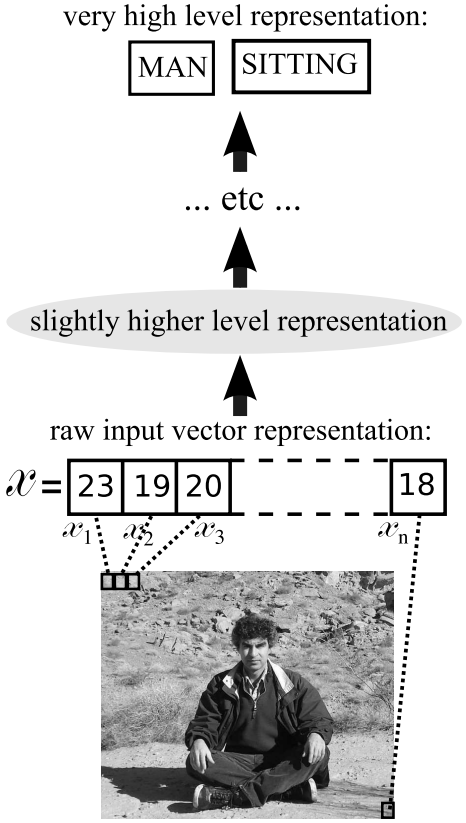

Figure 1: We would like the raw input image to be transformed into gradually higher levels of representation, representing more and more abstract functions of the raw input, e.g., edges, local shapes, object parts, etc. In practice, we do not know in advance what the "right" representation should be for all these levels of abstractions, although linguistic concepts might help us imagine what the higher levels might implicitly represent.

This paper has two main parts which can be read almost independently. In the first part, Sections 2, 3 and 4 use mathematical arguments to motivate deep architectures, in which each level is associated with a distributed representation of the input. The second part (in the remaining sections) covers current learning algorithms for deep architectures, with a focus on Deep Belief Networks, and their component layer, the Restricted Boltzmann Machine.

The next two sections of this paper review mathematical results that suggest limitations of many existing learning algorithms. Two aspects of these limitations are considered: insufficient *depth of architectures*, and *locality of estimators*. To understand 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 performed by an

artificial neuron. A function can be expressed by the composition of elements from this set, using 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 make (depending on its parameter values), depth corresponds to the number of layers in a neural network. Section 2 reviews theoretical results 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. This is detrimental for learning. Indeed, if a function represents a solution to the task 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. We say that the expression of a function is **compact** when it has few computational elements, i.e. less degrees of freedom that can be tuned by learning. So for a fixed number of training examples, we would expect that compact representations of the target function would yield better generalization.

Connected to the depth question is the question of locality of estimators, discussed in Section 3. This is another, more geometrically obvious, limitation of a large class of non-parametric learning algorithms: they obtain good generalization for a new input x by mostly exploiting training examples in the neighborhood of x. For example, the k nearest neighbors of the test point x, among the training examples, vote for the prediction at x. This locality 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.

Section 4 suggests how deep architectures could be exploited to extract multiple levels of **distributed representations**, where the set of configurations of values at each level of the computation graph can be very large. This would allow us to compactly represent a complicated function of the input.

In the remainder, the paper describes and analyses some of the algorithms that have been proposed to train deep architectures.[1] Many of these algorithms are based on the **autoassociator**: a simple unsupervised algorithm for learning a one-layer model that computes a distributed representation for its input (Rumelhart, Hinton, & Williams, 1986a; Bourlard & Kamp, 1988; Hinton & Zemel, 1994). We also discuss **convolutional neural networks**, the oldest successful example of deep architecture, specialized for vision and signal processing tasks (LeCun, Boser, Denker, Henderson, Howard, Hubbard, & Jackel, 1989; LeCun, Bottou, Bengio, & Haffner, 1998b). Sections 9 and 10 are devoted to a family of more recently proposed learning algorithms that have been very successful to train deep architectures: Deep Belief Networks (DBNs) (Hinton, Osindero, & Teh, 2006) and Stacked Autoassociators (Bengio, Lamblin, Popovici, & Larochelle, 2007; Ranzato, Poultney, Chopra, & LeCun, 2007). DBNs are based on Restricted Boltzmann Machines (RBMs) and the Contrastive Divergence algorithm (Hinton, 2002), introduced in Section 6. In Section 7 we describe estimators of the log-likelihood gradient for RBMs. This analysis shows how reconstruction error (used to train autoassociators), and Contrastive Divergence (used to train RBMs) approximate the log-likelihood gradient. Section 8 generalizes as much as possible the parametrization of RBMs so as to keep its basic factorizing property and the Contrastive Divergence estimator of the gradient. Finally, we consider the most challenging question: how can we possibly deal with the difficult optimization problem that training these deep architectures entails? This part of the paper contains mostly questions and suggestions for research directions. In particular, we discuss the principle of continuation methods, which first solves smoother versions of the desired cost function, to make a dent in the optimization of deep architectures, and we find that existing algorithms for RBMs and DBNs already are approximate continuation methods.

[1] Mostly deep neural networks, to date, but we suggest later that ensembles of trees could be learned and stacked similarly to layers in a neural network.

1.1 Desiderata for Learning AI

Summarizing some of the above issues, we state a number of requirements we perceive for learning algorithms to solve AI.

- 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 the "right" associated 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.
- In the limit of a large number of tasks and when future tasks are not known ahead of time, strong **unsupervised learning** (i.e. capturing the statistical structure in the observed data) is an important element of the solution.

Other elements are equally important but are not directly connected to the material in this paper. They include the ability to learn to represent context of varying length and structure (Pollack, 1990), so as to allow machines to operate in a stream of observations and produce a stream of actions, the ability to make decisions when actions influence the future observations and future rewards (Sutton & Barto, 1998), and the ability to influence future observations so as to collect more relevant information about the world (i.e. a form of active learning (Cohn, Ghahramani, & Jordan, 1995)).

2 Theoretical Limitations of Shallow Architectures

In this section, we present an argument in favor of deep architecture models by way of theoretical results revealing limitations of archictectures with insufficient depth. This part of the paper (this section and the next) motivate the algorithms described in the later sections, and can be skipped without making the remainder difficult to follow. The main conclusion of this section is that 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 just 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**.

Let us return to the notion of depth with more 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, and the $\sin$ operation, as illustrated in Figure 2. 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

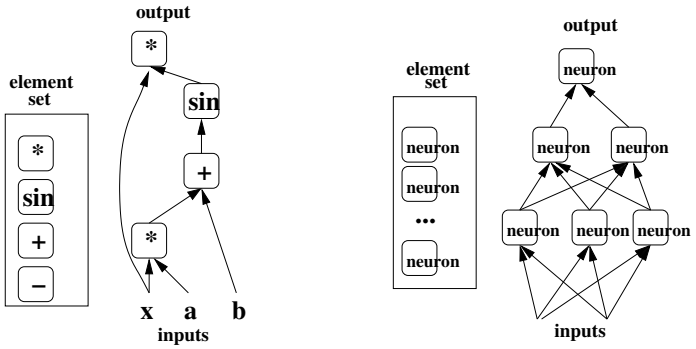

Figure 2: Examples of functions represented by a graph of computations, where each node is taken in some set of allowed computations. Left: the elements are $\{*, +, \sin\} \cup \mathbb{R}$. The architecture computes $x*\sin(a*x+b)$ and has depth 4. Right: the elements are artificial neurons computing $f(x) = \tanh(b+w'x)$; each element in the set has a different (w, b) parameter. The architecture is a multi-layer neural network of depth 3.

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. 3 in the case of $x * \sin(a * x + b)$ in Figure 2.

- If we include affine operations and 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 $K(u, v)$ in the set of allowed operations, along with affine operations, kernel machines (Schölkopf, Burges, & Smola, 1999a) with a fixed kernel can be considered to have two levels. The first level has one element computing $K(x, x_i)$ for each prototype x_i (a selected representative training example) and matches the input vector x with the prototypes x_i. The second level performs a linear combination $\sum_i \alpha_i K(x, x_i)$ to associate the matching prototypes 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 (Rumelhart et al., 1986a). 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 3.3.
- Boosting (Freund & Schapire, 1996) usually adds one level to its base learners: that level computes a vote or linear combination of the outputs of the base learners.
- Stacking (Wolpert, 1992) is another meta-learning algorithm that adds one level.
- Based on current knowledge of brain anatomy (Serre, Kreiman, Kouh, Cadieu, Knoblich, & Poggio, 2007), it appears that the cortex can be seen as a deep architecture, e.g., consider the many so-called layers in the visual system.

Although depth depends on the choice of the set of allowed computations for each element, 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). As we will describe, if a function can be compactly represented with k levels using a particular

choice of computational element set, it might require a huge number of computational elements to represent it with $k-1$ or less levels (using that same computational element set).
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 (Mendelson, 1997). 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 important result to consider is that with depth-two logical circuits, most boolean functions require an *exponential* number of logic gates (Wegener, 1987) to be represented (with respect to input size).
Furthermore, there are functions computable with a polynomial-size logic gates circuit of depth k that require exponential size when restricted to depth $k-1$ (Hastad, 1986). The proof of this theorem relies on earlier results (Yao, 1985) showing that *d-bit parity circuits of depth 2 have exponential size*. The d-bit **parity function** is defined as usual:

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

One might wonder whether these computational complexity results for boolean circuits are relevant to machine learning. See Orponen (1994) 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 (McCulloch & Pitts, 1943)), which compute

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

with parameters 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).
One might argue that the limitations of logic gates circuits might not apply to the kind of architectures found in machine learning algorithms. With that in mind, it is interesting to note that similar theorems were proved for circuits of linear threshold units, which are the computational elements of some multi-layer neural networks. 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 $N > N_0$ (Hastad & Goldmann, 1991).*

The class of functions $\mathcal{F}_{k,N}$ is defined as follows. It contains functions of N^{2k-2} variables each 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 i-th 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? 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.

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, whereas the polynomial $\prod_{i=1}^{n} \sum_{j=1}^{m} a_{ij} x_j$ can be represented efficiently as a product of sums, with only $O(mn)$ computational elements, it would be very inefficiently represented with a sum of product architecture, requiring $O(n^m)$ computational elements.

Further examples suggesting greater expressive power of deep architectures and their potential for AI and machine learning are also discussed in Bengio and Le Cun (2007). An earlier discussion of the expected advantages of deeper architectures in a more cognitive perspective is found in Utgoff and Stracuzzi (2002). Having established some theoretical grounds justifying the need for learning deep architectures, we next turn to a related question: deep architectures can represent highly-varying functions compactly, with less computational elements than there are variations in the represented function, but many state-of-the-art machine learning algorithms do not have that characteristic.

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 mean that depth of architecture can be very important from the point of view a statistical efficiency.

3 Local vs Non-Local Generalization: the Limits of Matching Local Templates

This section focuses on the locality of estimators in many shallow architectures, which gives rise to poor generalization when trying to learn highly-varying functions. This is because highly-varying functions, which can sometimes be represented efficiently with deep architectures, cannot be represented efficiently if the learning algorithm is a local estimator.

A **local estimator** partitions the input space in regions (possibly in a soft rather than hard way) and requires 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.

As an extreme example of a shallow and local architecture, consider a disjunctive normal form (depth 2) logic-gate circuit with all possible 2^n gates at the first level. The 2^n possibilities come from the choice, for each gate, of negating or not each of the n inputs before applying the AND computation. Each such product is called a **minterm**. One can see such as circuit simply as a very large pattern matcher. More generally, if only a subset of the input variables is used in a particular AND gate, then that gate will respond to a larger set of input patterns. The gate is then a template matcher that responds to patterns in a connected region of input space, e.g. the subspace that is the set of vectors x such that $x_1 = 1$, $x_2 = 0$ but x_3 and x_4 can take any value.

More generally, 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. The prototypical example of architectures based on maching local templates is the **kernel machine** (Schölkopf et al., 1999a)

$$f(x) = b + \sum_i \alpha_i K(x, x_i), \tag{2}$$

where b and α_i form the second level, **kernel function** $K(x, x_i)$ matches the input x to the training example x_i, and the sum runs over all or a subset of the input patterns of the training set. In the above equation, $f(x)$ could be the discriminant function of a classifier, or the output of regression predictor. A kernel is **local**, when $K(x, x_i) > \rho$ is true for x in some connected region around x_i. The size of that region can usually be controlled by a hyper-parameter. An example of local kernel is the Gaussian kernel $K(x, x_i) = e^{-||x-x_i||^2/\sigma^2}$, where σ controls the size of the region around 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: $K(u, v) = \prod_i e^{-(u_i-v_i)^2/\sigma^2}$. If $|u_i - v_i|/\sigma$ is small for all i, then the pattern matches and $K(u, v)$ is large. If $|u_i - v_i|/\sigma$ is large for a single i, then there is no match and $K(u, v)$ is small.

Well-known example of kernel machines include Support Vector Machines (SVMs) (Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995) and Gaussian processes (Williams & Rasmussen, 1996) [2] 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 and regression estimators, etc. In Section 3.2 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 (x_i, y_i), then it makes sense to construct a predictor $f(x)$ which will output something close to y_i when x is close to 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 in Bengio, Delalleau, and Le Roux (2006) and Bengio and Le Cun (2007) is that such a prior if often insufficient to generalize when the target function is highly-varying in input space (according to the notion of proximity embedded in the prior or kernel). Consider that most kernels used in practice can be seen as a dot product in a feature space: $K(x, x_i) = \phi(x) \cdot \phi(x_i)$, where generally $\phi(x)$ is a non-linear transformation of the input x into a high-dimensional **feature space**. A good feature space would be one where the target function is smooth when expressed in the feature space. One could therefore correctly argue that if the target function is highly varying in input space and in the kernel feature space, it might simply be because we have not selected the appropriate feature space. If our feature space does not have that property, i.e. the approximation $y \approx y_i$ when $\phi(x) \approx \phi(x_i)$ is only valid in a small region around $\phi(x_i)$, then one will need many such regions to cover the domain of interest. Unfortunately, at least as many training examples will be needed as there are regions necessary to cover the variations of interest in the target function.

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 (Jaakkola & Haussler, 1998; Schölkopf, Mika, Burges, Knirsch, Müller, Rätsch, & Smola, 1999b; Gärtner, 2003; Cortes, Haffner, & Mohri, 2004). However, if we lack sufficient prior knowledge for designing an appropriate kernel, can we learn it? this question also motivated much research (Lanckriet, Cristianini, Bartlett, El Gahoui, & Jordan, 2002; Wang & Luk Chan, 2002; Cristianini, Shawe-Taylor, Elisseeff, & Kandola, 2002), 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 (Salakhutdinov & Hinton, 2008): predictions are improved by using the top-level representation instead of the raw input representation, and they are further improved by tuning the deep network to minimize the prediction error made by

[2] In the Gaussian Process case, as in kernel regression, $f(x)$ in eq. 2 is the conditional expectation of the target variable Y to predict, given the input x

the Gaussian process, using gradients of the prediction error back-propagated into the neural network. The feature space can be seen as a representation of the data. Good representations make examples which share abstract characteristics close to each other. Learning algorithms for deep architectures can be seen as ways to learn a good feature space for kernel machines.
In the next subsection we review theoretical results on the limitations of kernel machines with a Gaussian kernel in the case of supervised learning, which show that the required number of examples grows linearly with the number of bumps in the target function to be learned. In subsection 3.2 we present results of a similar flavor for semi-supervised non-parametric learning algorithms, and in subsection 3.3 for decision trees. We conclude in subsection 3.4 with a discussion on the use of smoothness as a prior, and how it can be made more powerful by extending the notion of complexity of a function, in the extreme case using Kolmogorov complexity.

3.1 Theoretical Limitations of Local Kernels

Here we consider formal results about limitations of local kernel machines. The notion that local kernels are insufficient to capture highly-varying functions is formalized in a few particular cases in Bengio et al. (2006), Bengio and Le Cun (2007). One result is the following:

Theorem 3.1. *Suppose that the learning problem is such that in order to achieve with a Gaussian kernel machine (eq. 2) a given error level for samples from a distribution P, f must change sign at least $2k$ times along some straight line (i.e., in the case of a classifier, a sufficiently good decision surface must be crossed at least $2k$ times by that straight line). Then the kernel machine must have at least k bases (non-zero α_i's), and hence at least k training examples.*

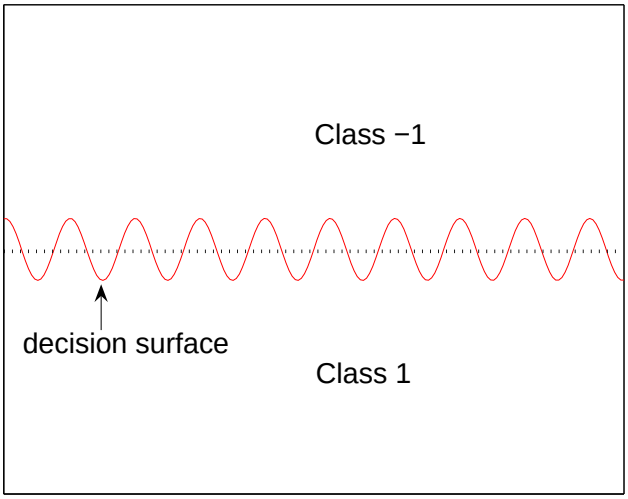

Figure 3: The dotted line crosses the decision surface 19 times: according to Theorem 3.1, and in line with intuition, one needs 10 Gaussians to learn it with an affine combination of Gaussians, with each Gaussian capturing one of the bumps in the function.

This theorem says that we need as many examples as there are variations ("bumps") in the function that we wish to represent with a Gaussian kernel machine. As illustrated in Figure 3, a function may have a

large number of variations (e.g. a sinusoidal) and yet be representable much more compactly because these variations are interdependent. It is conceivable that a different learning algorithm could take advantage of the global regularity (repeating pattern) to learn it with few parameters (thus requiring few examples). By contrast, with an affine combination of Gaussians, theorem 3.1 implies one would need at least $\lceil \frac{m}{2} \rceil = 10$ Gaussians. With a local estimator, it is plausible that more examples will be needed to take care of new instances of the repeating pattern in the curve. 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 local estimators. However, it might be worth it to look 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, but these are the functions that we want to learn for AI.
A different type of variability is illustrated by the parity function, where a small change in any direction in input space corresponds to a large change in the desired output. In that case one can show (Bengio et al., 2006) that the number of examples necessary with a Gaussian kernel machine is exponential in the input dimension:

Theorem 3.2. *Let $f(x) = b + \sum_{i=1}^{2^d} \alpha_i K_\sigma(x_i, x)$ be an affine combination of Gaussians with same width σ centered on points $x_i \in \{-1, 1\}^d$. If f solves the parity problem, then there are at least 2^{d-1} non-zero coefficients α_i.*

Note that one way in which the parity function is not representative of the kind of functions we are more interested in AI is that the target function does not depend on the order of the inputs. Also, parity can be represented with a shallow neural network with $O(d)$ units (and $O(d^2)$ parameters). This solution exploits the fact that projecting the input vector x to the scalar $s = \sum_i x_i$ preserves the information that is necessary to compute parity: it is enough to consider in which of $d+1$ intervals s falls to determine the correct answer, and d threshold units are sufficient to achieve this.
Hence the theoretical results discussed in this section are merely suggestive but do not prove that the learning algorithms for the functions that we need to represent for AI should not be local estimators.

3.2 Unsupervised and Semi-Supervised Algorithms Based on Neighborhood-Graph

Local estimators are found not only in supervised learning algorithms such as those discussed above, but also in unsupervised and semi-supervised learning algorithms, to which we now turn. Here again, we find that in order to 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.
Unsupervised learning algorithms attempt to capture characteristics of the input distribution. For example, **manifold learning** algorithms attempt to discover a lower-dimensional region near which the density concentrates. There is a connection between kernel machines such as SVMs and Gaussian processes and a number of unsupervised and semi-supervised learning algorithms: many of these unsupervised and semi-supervised algorithms can be expressed as kernel machines with a particular kernel, one that is possibly data-dependent (Bengio, Delalleau, Le Roux, Paiement, Vincent, & Ouimet, 2004). The following unsupervised learning algorithms, included in this analysis, attempt to capture the manifold structure of the data by capturing its local changes in shape: Locally Linear Embedding (Roweis & Saul, 2000), Isomap (Tenenbaum, de Silva, & Langford, 2000), kernel Principal Components Analysis (Schölkopf, Smola, & Müller, 1998) (or kernel PCA) Laplacian Eigenmaps (Belkin & Niyogi, 2003), Manifold Charting (Brand, 2003), and spectral clustering algorithms (see Weiss (1999) for a review). Several non-parametric semi-supervised learning algorithms are based on similar concepts, involving the use of a kernel (Zhu, Ghahramani, & Lafferty, 2003; Zhou, Bousquet, Navin Lal, Weston, & Schölkopf, 2004; Belkin, Matveeva, & Niyogi, 2004; Delalleau, Bengio, & Le Roux, 2005).

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. The question we want to discuss here is whether the above non-parametric algorithms are likely to suffer from the same limitations already discussed for local kernel machines for classification or regression in the previous section. With these algorithms, one can get geometric intuition of what they are doing, as well as how being a local estimators can hinder them. This is illustrated with the example in Figure 4 in the case of manifold learning. The issue is related to the curse of dimensionality: to cover all the variations with locally linear patches, a lot of patches might be necessary, and enough examples in each patch to characterize its shape, i.e. the tangent plane at the patch location.

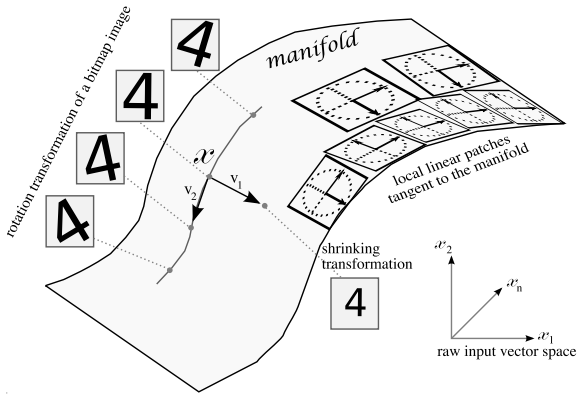

Figure 4: The set of images associated with the same object class forms a manifold, i.e. a region of lower dimension than the original space of images. By rotating, translating, or shrinking an image, e.g., of digit 4, we get other images of the same class, i.e. on the same manifold. Since the manifold 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.

Similar limitations have been proved for a large class of semi-supervised learning algorithms also based on the neighborhood graph (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2004; Delalleau et al., 2005). 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 (Bengio et al., 2006). 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.

3.3 Decision Trees Do not Generalize to New Variations

Decision trees 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 (Bengio, Delalleau, & Simard, 2007). As we argue here, this means that they also suffer from the limitation discussed for other non-parametric learning algorithms in the previous sections: 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.

As illustrated in Figure 5, a decision tree recursively partitions the input space and assigns an output value for each of the input regions in that partition. Learning algorithms for decision trees (Breiman, Friedman, Olshen, & Stone, 1984) are non-parametric and involve a non-convex optimization to choose a tree structure and parameters associated with nodes and leaves. Fortunately, greedy heuristics that build the tree incrementally have been found to work well. Each node of the tree corresponds to a region of the input space,

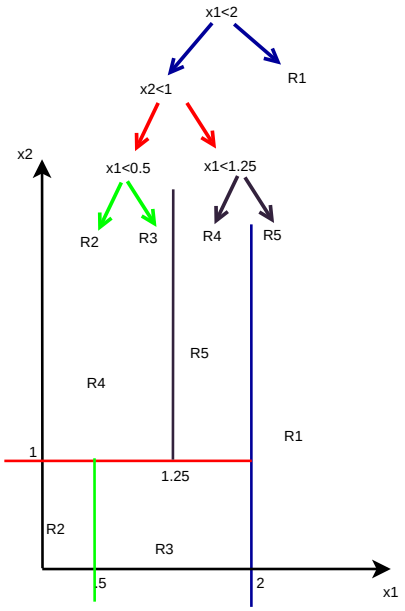

Figure 5: A decision tree recursively partitions the input space. In a binary tree, the root node splits it in two. Each node is associated with a region. An output value is learned for each leaf node region.

and the root is associated with the whole input space. We call **constant-leaves decision tree** (the common type) one where the whole tree corresponds to a piece-wise constant function where the pieces are defined by the internal decision nodes: each leaf is associated with one piece, along with a constant to output in the associated region. The decision nodes on the path from the root to a leaf define one of the mutually exclusive regions formed by the decision tree. Like in a disjunctive normal form circuit or a Gaussian kernel machine, the outputs of decision nodes are multiplied and form a conjunction: an example has to satisfy all the conditions to belong to a leaf region. The decision nodes form the first level of the architecture. The predictions associated with the leaves, along with their parameters, form the second level of the architecture. Bengio et al. (2007) study fundamental limitations of decision trees concerning their inability to *generalize to variations not seen in the training set.* The basic argument is that a decision tree needs a separate leaf node to properly model each such variation, and at least one training example for each leaf node. That theoretical analysis is built along lines similar to ideas exploited previously in the computational complexity literature (Cucker & Grigoriev, 1999). These results are also in line with previous empirical results (Pérez & Rendell, 1996; Vilalta, Blix, & Rendell, 1997) showing that the generalization performance of decision trees degrades when the number of variations in the target function increases.
The following results are taken from Bengio et al. (2007).

Proposition 3.3. *Let $\mathcal{F}$ be the set of piece-wise constant functions. Consider a target function $h : \mathbb{R}^d \rightarrow \mathbb{R}$. For a given representation error level ϵ, let N be the minimum number of constant pieces required to approximate, with a function in $\mathcal{F}$, the target function h with an error less than ϵ. Then to train a constant-leaves decision tree with error less than ϵ one requires at least N training examples.*

The above proposition states that the number of examples needed grows linearly with the number of regions needed to achieve a desired error level. The theorem below states a more specific result in the case of a family of function for which the number of needed regions is exponential in the input size.

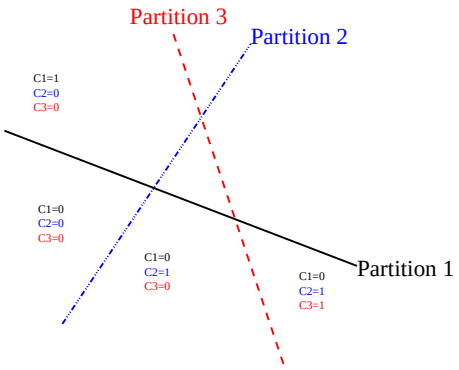

Figure 6: Whereas a single tree can discriminate among a number of regions linear in the number of parameters (leaves), an ensemble of trees can discriminate among a number of regions exponential in the number of trees, i.e. exponential in the total number of parameters (at least as long as the number of trees is less or equal to the number of inputs). Each distinguishable region is associated with one of the leaves of each tree (here there are 3 trees, each defining 2 regions, for a total of 7 regions). This is equivalent to a multi-clustering, here 3 clusterings each associated with 2 regions. A binomial RBM is a multi-clustering with 2 linearly separated regions per partition (each associated with one hidden unit). A multi-clustering is therefore a distributed representation of the input pattern.

Theorem 3.4. *On the task of learning the d-bit parity function, a constant-leaves decision tree with axis-aligned decision nodes will require at least $2^d(1-2\epsilon)$ examples in order to achieve a generalization error less than or equal to ϵ.*

Ensembles of trees (like boosted trees (Freund & Schapire, 1996), and forests (Ho, 1995; Breiman, 2001)) 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* (Bengio et al., 2007). As illustrated in Figure 6, they implicitly form a *distributed representation* (a notion discussed further in Section 4) 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 that tree. The description of an input pattern with the identities of the leaf nodes for the trees is very rich: 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. Since a depth $k-1$ architecture might be very inefficient to represent a depth k function, it might be interesting to explore learning algorithms based upon decision trees in which the architecture depth is even greater than in ensembles of trees.

3.4 Smoothness versus Kolmogorov Complexity

To escape the curse of dimensionality, it is necessary to have a model that can capture a large number of variations that can occur in the data without having to enumerate all of them. Instead, a compact representation that captures most of these variations has to be discovered by the learning algorithm. Here "compact" means that it *could be* encoded with a few bits.

The notion of local estimator is connected to the notion of smoothness and smoothness priors introduced at the beginning of Section 3. Smoothness as a measure of simplicity is a useful way to control generalization, but others are possible, and probably more desirable. For example, consider a target function is highly-varying with a number of variations much larger than the number of training examples one can hope to get. A deep architecture could potentially represent such a function with a small number of parameters (comparable to the number of training examples one could get). If one discovers such a compact representation of the target function, then a form of compression has been achieved. This is likely to yield good

generalization (Solomonoff, 1964; Kolmogorov, 1965; Li & Vitanyi, 1997; Hutter, 2005) because of Occam's Razor. Maybe the most extreme and general way to measure that compression is with Kolmogorov complexity. The **Kolmogorov complexity** is the length of the smallest string that represents the solution, in some programming language. Using a different language only adds a constant to the string length (for the code that translates strings in one language to strings in another). It is clear that many functions expressible with a very short string can be highly varying, such as the sinus example of Figure 3. Learning theory (Vapnik, 1995; Li & Vitanyi, 1997) shows that if a compact description can be found to summarize the training set, good generalization is to be expected.

The main advantage of smoothness expressed through a kernel or covariance function (in Gaussian processes) is that the optimization problem involved in the learning algorithm can be convex, i.e. devoid of local minima and hence easy to solve. Kolmogorov complexity is not even computable, but it can be bounded from above. Upper bounds on Kolmogorov complexity can be optimized. Our thesis is that deep architectures can represent many functions compactly, and that their approximate optimization might yield very good solutions even if the global optimum is not found: any solution that is more compact than previous ones brings a gain in generalization. Minimum Description Length (Rissanen, 1990) and its variants such as Minimum Message Length (Wallace & Boulton, 1968) also use this principle in the context of random variables with many realizations: a good predictive model (in terms of out-of-sample log-likelihood) is also one that can assign a short code to each example, *on average*, including not only the bits to describe each example but also the bits necessary to describe the model itself.

What can be concluded from our analysis of limitations of learning algorithms due to insufficient depth and local estimation? In either case, insufficient depth or local estimator, we found that one might need to represent the target function with a very large number of tunable elements, and thus one would need a very large number of examples. On the other hand, if a representation exists that can compactly represent the target function, then good generalization could be obtained from a number of examples much smaller than the number of variations of the target function. An important idea that gives hope of compactly representing a very large number of configurations is the idea of distributed representation, discussed next, and which introduces the second part of this paper, about learning algorithms for deep architectures.

4 Learning Distributed Representations

An old idea in machine learning and neural networks research, which could be of help in dealing with the curse of dimensionality and the limitations of local generalization is that of **distributed representations** (Hinton, 1986; Rumelhart, McClelland, & the PDP Research Group, 1986b; Bengio, Ducharme, & Vincent, 2001). A cartoon **local representation** for integers $i \in \{1, 2, \ldots, N\}$ is a vector $r(i)$ of N bits with a single 1 and $N-1$ zeros, $r_j(i) = \mathbb{1}_{i=j}$, called the **one-hot** representation of i. A distributed representation for the same integer is 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. In practice, we use local representations which are continuous-valued vectors where the i-th element varies according to some distance between the input and a prototype or region center, as with the Gaussian kernel discussed in Section 3. 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 Components Analysis (Bell & Sejnowski, 1995; Pearlmutter & Parra, 1996) and Principal Components Analysis or PCA (Hotelling, 1933) build a distributed representation.

Consider a discrete distributed representation $r(x)$ for an input pattern x, where $r_i(x) \in \{0, 1, \ldots M\}$, $i \in \{1, \ldots, N\}$. Each $r_i(x)$ can be seen as a classification of x into M classes. Each $r_i(x)$ partitions the x-space in M regions, but the different partitions can be combined to give rise to a potentially exponential number of possible regions in x-space, corresponding to different configurations of $r_i(x)$. Note that 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 a number of 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?). Like in clustering, we construct discrete classes, but the potential number of combined classes is huge: we obtain what we call a **multi-clustering**. Whereas clustering forms a single partition and generally involves a loss of information about the input, a multi-clustering provides a *set* of separate partitions of the input space. Identifying to which region of each partition the input example belongs forms a description of the input pattern which might be very rich, possibly not losing any information. The tuple of symbols specifying to which region of each partition the input belongs 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 x-space that an ensemble of trees can represent, as discussed in the previous section.

In the realm of supervised learning, multi-layer neural networks (Rumelhart et al., 1986b, 1986a) and Boltzmann machines (Ackley, Hinton, & Sejnowski, 1985) 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 optimization problem, which is central in the remainder of this paper.

5 Learning Deep Architectures: a Difficult Optimization Problem

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 involves optimization problems that are more difficult than those involved in training shallow architectures (Bengio et al., 2007). Much of that evidence comes from research on training multi-layer neural networks, suggesting that training gets stuck in local minima or plateaus, with worse results than with neural networks with one or two hidden layers.

A typical set of equations for multi-layer neural networks is the following. As illustrated in Figure 7, layer ℓ computes an output vector $\mathbf{z}_\ell$ using the output $\mathbf{z}_{\ell-1}$ of the previous layer, starting with the input $\mathbf{z}_0$,

$$\mathbf{z}_\ell = \tanh(b_\ell + W_\ell \mathbf{z}_{\ell-1}) \tag{3}$$

with parameters b_ℓ (the biases) and W_ℓ (the weights). The $\tanh$ can be replaced by $\mathrm{sigm}(x) = 1/(1+e^{-x}) = \frac{1}{2}(\tanh(x)+1)$. The top layer output $\mathbf{z}_L$ is used for making a prediction and is combined with a supervised target y into a loss function $L(\mathbf{z}_L, y)$, typically convex. The output layer might have a non-linearity different from $\tanh$, e.g., the softmax

$$z_{Li} = \frac{e^{b_{Li}+W_{Li}\mathbf{z}_{L-1}}}{\sum_j e^{b_{Lj}+W_{Lj}\mathbf{z}_{L-1}}} \tag{4}$$

where W_{Li} is the i-th row of W_L, z_{Li} is positive and $\sum_i z_{Li} = 1$. The softmax output z_{Li} can be used as estimator of $P(Y = i|x)$, with the interpretation that $Y = i$ is the i-th class associated with input pattern x. In this case one often uses the negative conditional log-likelihood $L(z_L, y) = -\log z_{Ly}$ as a loss, whose expected value over (x, y) pairs is to be minimized.

Deep architectures have not been studied much in the machine learning literature, because of the difficulty in optimizing them (Bengio et al., 2007). Notable exceptions include **convolutional neural networks** (LeCun et al., 1989; LeCun et al., 1998b; Simard & Platt, 2003; Ranzato et al., 2007), and Sigmoidal Belief Networks using variational approximations (Dayan, Hinton, Neal, & Zemel, 1995; Hinton, Dayan, Frey, & Neal, 1995; Saul, Jaakkola, & Jordan, 1996; Titov & Henderson, 2007), and more recently Deep Belief Networks (Hinton et al., 2006; Bengio et al., 2007). Many unreported negative observations as well as the experimental results

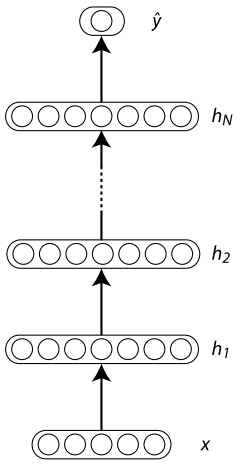

Figure 7: Multi-layer neural network, typically used in supervised learning to make a prediction or classification, through a series of layers each of which combines an affine operation and a non-linearity. Computations are performed in a feedforward way from the input x, through the hidden layers h_k, to the network output $\hat{y}$, which gets compared with a label y to obtain the loss $L(\hat{y}, y)$ to be minimized.

in Bengio et al. (2007) suggest that gradient-based training of deep supervised multi-layer neural networks gets stuck in local minima or plateaus. These appear to correspond to poor solutions that perform worse than the solutions obtained for networks with 1 or 2 hidden layers. The experiments reported in Bengio et al. (2007) also help to disantangle the effect of poor optimization with the effect of overfitting. They reveal that in a gradient-trained deep supervised neural network with random parameter initialization, the lower layers (closer to inputs) are poorly optimized. Indeed, we know that a two-layer network 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. If the top layers have enough capacity (enough hidden units) this can be sufficient to bring training error very low, but this yields worse generalization than shallow neural networks. On the other hand, with better initialization of the lower hidden layers, both training and generalization error can be very low. 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 hurt rather than help the top layers to perform classification. It is simple to obtain very small *training error* by simply increasing the capacity of the top layer(s). For example, optimizing the last layer of a deep neural network is usually a convex optimization problem. Optimizing the last two layers, although not convex, is known to be much easier than optimizing a deep network. Hence, what matters for good generalization, and is more difficult, is the optimization of the lower layers (excluding the last one or two). These are the layers that can give rise to a good representation of the input, in the sense that better generalization can be achieved from these representations. We believe that good representations capture the factors of variation in the input space and also disentangle them.

Although replacing the top two layers of a deep neural network by a convex machine such as a Gaussian process or an SVM can yield some improvements (Bengio & Le Cun, 2007), especially on the training error, it won't help much in terms of generalization if the lower layers have not been sufficiently optimized.

The above clues suggest that the gradient propagated backwards into the lower layer is not sufficient to move the parameters into regions corresponding to good solutions. Basically the optimization gets stuck in a poor local minimum or plateau (i.e. small gradient). Since gradient-based training of the top layers works reasonably well, it appears that the gradient becomes less informative about the required changes in the parameters as we move towards the lower layers. There might be some connection between this

difficulty in exploiting the gradient and the difficulty in training recurrent neural networks through long sequences, analyzed in (Hochreiter, 1991; Bengio, Simard, & Frasconi, 1994; Lin, Horne, Tino, & Giles, 1995). In recurrent neural networks, the difficulty can be traced to a vanishing (or sometimes explosion) of the 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 and long-term components of the gradient.

5.1 Convolutional Neural Networks

Although deep neural networks were generally found too difficult to train well, 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 Hubel and Wiesel (1962). 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 (Fukushima, 1980). 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 followed-up on this idea and trained such networks using the error gradient, obtaining and maintaining state-of-the-art performances (LeCun et al., 1989; LeCun et al., 1998b) on several vision tasks. Modern understanding of the physiology of the visual system is consistent with the processing style found convolutional networks (Serre et al., 2007), at least for the quick recognition of objects, i.e., without the benefit of attention and top-down feedback connections. To this day, vision systems based on convolutional neural networks are among the best performing systems. This has been shown clearly for handwritten character recognition (LeCun et al., 1998b), which has served as a machine learning benchmark for many years.[3]

Concerning our discussion of training deep architecturs, the example of convolutional neural networks (LeCun et al., 1989; LeCun et al., 1998b; Simard & Platt, 2003; Ranzato et al., 2007) is interesting because they typically have five, six or seven layers, a number of layers which makes fully-connected neural networks almost impossible to optimize properly when initialized randomly. What is particular in their 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 weights, associated with neurons in a rectangular patch in the previous layer. The same set of weights, but with a different input rectangular patch, is associated with neurons at different locations.

One untested hypothesis is that the small fan-in of these neurons (few inputs per neuron) allows gradients to propagate through so many layers without diffusing so much as to become useless. That 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 networks performs well (Ranzato, Huang, Boureau, & LeCun, 2007), i.e., better than a trained fully connected neural network but worse than a fully optimized convolutional neural network.

[3] Maybe too many years? It is good that the field is moving towards more ambitious benchmarks, such as those introduced in Larochelle, Erhan, Courville, Bergstra, and Bengio (2007).

5.2 Autoassociators

Some of the deep architectures discussed below (Deep Belief Nets and stacked autoassociators) exploit as component or monitoring device a particular type of neural network: the autoassociator, also called auto-encoder, or Diabolo network (Rumelhart et al., 1986a; Bourlard & Kamp, 1988; Hinton & Zemel, 1994; Schwenk & Milgram, 1995; Japkowicz, Hanson, & Gluck, 2000). There are also connections between the autoassociator and RBMs discussed in Section 7. Because training an autoassociator seems easier than training a deep network, they have been used as building blocks to train deep networks, where each level is associated with an autoassociator that can be trained separately.

An autoassociator is trained to encode the input in some representation so that the input can be reconstructed from that representation. Hence the target output is the 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 (Bourlard & Kamp, 1988). If the hidden layer is non-linear, the autoassociator behaves very differently from PCA, with the ability to capture multi-modal aspects of the input distribution (Japkowicz et al., 2000). 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 $c(x)$:

$$RE = -\log P(x|c(x)). \tag{5}$$

For example, if the inputs x_i are either binary or considered to be binomial probabilities, then the loss function would be

$$-\log P(x|c(x)) = -\sum_i x_i \log f_i(c(x)) + (1 - x_i) \log(1 - f_i(c(x))) \tag{6}$$

where $f(c(x))$ is the output of 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 $c(x)$ is a distributed representation that captures the main factors of variation in the data.

One serious issue with this approach is that if there is no other constraint, then an autoassociator with n-dimensional input and an encoding of dimension greater or equal to 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 (Bengio et al., 2007) suggest that in practice, when trained with stochastic gradient descent, autoassociators with more hidden units than inputs 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 $\ell 2$ regularization of the parameters (Collobert & Bengio, 2004). To achieve perfect reconstruction of continuous inputs, a one-hidden layer autoassociator 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 and very small 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 find 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 approximate the identity through a function and its inverse.

Instead or in addition to constraining the encoding by explicit or implicit regularization, 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 (Olshausen & Field, 1997; Doi, Balcan, & Lewicki, 2006; Ranzato et al., 2007; Ranzato & LeCun, 2007; Ranzato, Boureau, & LeCun, 2008), 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, a major area of the mammal visual system. The question of sparsity is discussed further in Section 13.2.

5.3 Unsupervised Learning as an Optimization Strategy

Another principle that has been found to help optimizing deep networks is based on the use of unsupervised learning to initialize each layer in the network. If gradients with respect to 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 expects this if the single-layer learning algorithm discovered a representation that captures statistical regularities of the layer's input. PCA and most variants of ICA seem inappropriate because they generally do not make sense in the so-called **overcomplete case**, where the number of outputs of the layer is is greater than the number of inputs of the layer. This suggests looking in the direction of extensions of ICA to deal with the overcomplete case (Lewicki & Sejnowski, 1998; Hinton, Welling, Teh, & Osindero, 2001; Teh, Welling, Osindero, & Hinton, 2003), as well as algorithms related to PCA and ICA, such as autoassociators and Restricted Boltzmann Machines, 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 (Hinton et al., 2006; Bengio et al., 2007; Ranzato et al., 2007).

In addition to the motivation that unsupervised learning could help reduce the dependency on the unreliable update direction given by the gradient with respect to a supervised criterion, there is 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 formalize the concepts behind RBMs.

6 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).

6.1 Energy-Based Models and Products of Experts

Energy-based models associate a scalar energy to each configuration of the variables of interest (LeCun & Huang, 2005; LeCun, Chopra, Hadsell, Ranzato, & Huang, 2006; Ranzato, Boureau, Chopra, & LeCun, 2007). 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 define a probability distribution through an energy function, as follows:

$$P(x) = \frac{e^{-\text{Energy}(x)}}{Z}. \tag{7}$$

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

$$Z = \sum_x e^{-\text{Energy}(x)} \tag{8}$$

with a sum running over the input space, or an appropriate integral when x is continuous.
In the **products of experts** formulation (Hinton, 1999, 2002), the energy function is a sum of terms, each one associated with an "expert" f_i:

$$\text{Energy}(x) = \sum_i f_i(x), \tag{9}$$

i.e.

$$P(x) \propto= P_i(x) \propto \prod_i e^{-f_i(x)} \tag{10}$$

Each expert $P_i(x)$ can thus be seen as a detector of implausible configurations of x, or equivalently, as enforcing constraints on x. This is clearer if we consider the special case where $f_i(x)$ 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. Hinton (1999) 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 $f_i(x)$ 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). Hinton (1999) proposed an algorithm for estimating the gradient of $\log P(x)$ in eq. 10 with respect to parameters associated with each expert, using a variant (Hinton, 2002) of the Contrastive Divergence algorithm described below.

6.1.1 Introducing Hidden Variables

In many cases of interest, we do not observe the example x fully, 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 x) and a *hidden* part h

$$P(x, h) = \frac{e^{-\text{Energy}(x,h)}}{Z} \tag{11}$$

and because only x is observed, we care about the marginal

$$P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z}. \tag{12}$$

In such cases, to map this formulation to one similar to eq. 7, we introduce the notation (inspired from physics) of **free energy**, defined as follows:

$$P(x) = \frac{e^{-\text{FreeEnergy}(x)}}{\sum_x e^{-\text{FreeEnergy}(x)}}, \tag{13}$$

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

$$\text{FreeEnergy}(x) = -\log \sum_h e^{-\text{Energy}(x,h)}. \tag{14}$$

The data log-likelihood gradient then has a particularly interesting form. Let us introduce θ to represent parameters of the model. Starting from eq. 13, we obtain

$$\begin{aligned} \frac{\partial \log P(x)}{\partial \theta} &= -\frac{\partial \text{FreeEnergy}(x)}{\partial \theta} + \frac{1}{Z} \sum_{\tilde{x}} e^{-\text{FreeEnergy}(\tilde{x})} \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta} \\ &= -\frac{\partial \text{FreeEnergy}(x)}{\partial \theta} + \sum_{\tilde{x}} P(\tilde{x}) \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta}. \end{aligned} \tag{15}$$

Hence the average log-likelihood gradient is

$$E_{\hat{P}}\left[\frac{\partial \log P(x)}{\partial \theta}\right] = -E_{\hat{P}}\left[\frac{\partial \text{FreeEnergy}(x)}{\partial \theta}\right] + E_P\left[\frac{\partial \text{FreeEnergy}(x)}{\partial \theta}\right] \tag{16}$$

where $\hat{P}$ denotes the training set empirical distribution and E_P denotes expected value 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 way to obtain a stochastic estimator of the log-likelihood gradient.
If the energy can be written as a sum of terms associated with each hidden unit (or to none)

$$\text{Energy}(x, h) = -\beta(x) + \sum_i \gamma_i(x, h_i), \tag{17}$$

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):

$$\begin{aligned}
P(x) &= \frac{1}{Z} e^{-\text{FreeEnergy}(x)} = \frac{1}{Z} \sum_h e^{-\text{Energy}(x,h)} \\
&= \frac{1}{Z} \sum_{h_1} \sum_{h_2} \cdots \sum_{h_k} e^{\beta(x) - \sum_i \gamma_i(x,h_i)} = \frac{1}{Z} \sum_{h_1} \sum_{h_2} \cdots \sum_{h_k} e^{\beta(x)} \prod_i e^{-\gamma_i(x,h_i)} \\
&= \frac{e^{\beta(x)}}{Z} \sum_{h_1} e^{-\gamma_1(x,h_1)} \sum_{h_2} e^{-\gamma_2(x,h_2)} \cdots \sum_{h_k} e^{-\gamma_k(x,h_k)} \\
&= \frac{e^{\beta(x)}}{Z} \prod_i \sum_{h_i} e^{-\gamma_i(x,h_i)}
\end{aligned} \tag{18}$$

In the above, $\sum_{h_i}$ is a sum over all the values that h_i can take. Note that all sums can be replaced by integrals if 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. the free energy) can be computed exactly in the case where $\text{Energy}(x, h) = -\beta(x) + \sum_i \gamma_i(x, h_i)$, and we have

$$\text{FreeEnergy}(x) = -\beta(x) - \sum_i \log \sum_{h_i} e^{-\gamma_i(x,h_i)}. \tag{19}$$

6.1.2 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 x, instead of considering all configurations (x, y), it is enough to consider the configurations of y for each given x. A common case is one where y can only take values in a small discrete set, i.e.

$$P(y|x) = \frac{e^{-\text{Energy}(x,y)}}{\sum_y e^{-\text{Energy}(x,y)}}. \tag{20}$$

In this case the gradient of the conditional log-likelihood with respect to parameters of the energy function can be computed efficiently. This approach has been exploited in a series of probabilistic language models based on neural networks (Bengio et al., 2001; Schwenk & Gauvain, 2002; Bengio, Ducharme, Vincent, & Jauvin, 2003; Xu, Emami, & Jelinek, 2003; Schwenk, 2004; Schwenk & Gauvain, 2005). That formulation (or generally when it is easy to sum or maximize over the set of values of the terms of partition function) has been explored at length (LeCun & Huang, 2005; LeCun et al., 2006; Ranzato et al., 2007, 2007). 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. This criterion does not necessarily give rise to a probabilistic model, but it gives rise to a function that can be used to choose y given x, which is often the ultimate goal in applications.

6.2 Boltzmann Machines

The Boltzmann machine is a particular type of energy-based model, and RBMs are special forms of Boltzmann machines in which $P(h|x)$ and $P(x|h)$ are both tractable because they factorize. In a Boltzmann machine (Hinton, Sejnowski, & Ackley, 1984; Ackley et al., 1985; Hinton & Sejnowski, 1986), the energy function is a second-order polynomial:

$$\text{Energy}(x, h) = -b'x - c'h - h'Wx - x'Ux - h'Vh. \tag{21}$$

There are two types of parameters, which we collectively denote by θ: the biases b_i and c_i (each associated with a single element of the vector x or of the vector 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, 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 (Welling, Rosen-Zvi, & Hinton, 2005).
Because of the quadratic interaction terms in h, the trick to analytically compute the free energy (eq. 18) cannot be applied here. However, an MCMC (Monte Carlo Markov Chain (Andrieu, de Freitas, Doucet, & Jordan, 2003)) 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 eq. 12:

$$\begin{aligned}
\frac{\partial \log P(x)}{\partial \theta} &= \frac{\partial \log \sum_h e^{-\text{Energy}(x,h)}}{\partial \theta} - \frac{\partial \log \sum_{x,h} e^{-\text{Energy}(x,h)}}{\partial \theta} \\
&= -\frac{1}{\sum_h e^{-\text{Energy}(x,h)}} \sum_h e^{-\text{Energy}(x,h)} \frac{\partial \text{Energy}(x,h)}{\partial \theta} \\
&\quad + \frac{1}{\sum_{x,h} e^{-\text{Energy}(x,h)}} \sum_{x,h} e^{-\text{Energy}(x,h)} \frac{\partial \text{Energy}(x,h)}{\partial \theta} \\
&= -\sum_h P(h|x) \frac{\partial \text{Energy}(x,h)}{\partial \theta} + \sum_{x,h} P(x,h) \frac{\partial \text{Energy}(x,h)}{\partial \theta}.
\end{aligned} \tag{22}$$

Note that $\frac{\partial \text{Energy}(x,h)}{\partial \theta}$ is easy to compute. Hence if we have a procedure to sample from $P(h|x)$ and one to sample from $P(x, h)$, we can obtain an unbiased stochastic estimator of the log-likelihood gradient. Hinton et al. (1984), Ackley et al. (1985), Hinton and Sejnowski (1986) introduced the following terminology: in the **positive phase**, x is **clamped** to the observed input vector, and we sample h given x; in the **negative phase** both x and h are sampled, ideally from the model itself. Only approximate sampling can be achieved tractably, e.g., using an iterative procedure that constructs an MCMC. The MCMC sampling approach introduced in Hinton et al. (1984), Ackley et al. (1985), Hinton and Sejnowski (1986) is based on **Gibbs sampling** (Geman & Geman, 1984; Andrieu et al., 2003). Gibbs sampling of the joint of N random variables $X_1 \ldots X_N$ is done through a sequence of N sampling sub-steps of the form

$$X_i \sim P(X_i | X_{-i} = x_{-i}) \tag{23}$$

where X_{-i} contains the $N-1$ other random variables in X, excluding X_i. After these N samples have been obtained, a step of the chain is completed, yielding a sample of X whose distribution converges to $P(X)$ as the number of steps goes to ∞.
Let $y = (x, h)$ denote all the units in the Boltzmann machine, and y_{-i} the set of values associated with all units except the i-th one. The Boltzmann machine energy function can be rewritten by putting all the parameters in a vector d and a symmetric matrix A,

$$\text{Energy}(y) = -d'y - y'Ay, \tag{24}$$

with d_{-i} the vector d without the element d_i, A_{-i} the matrix A without the i-th row and column, and A_i the vector that is the i-th row (or column) of A, without the i-th element. The idea is to exploit the fact that

$P(y_i|y_{-i})$ can be computed and sampled from easily in a Boltzmann machine. For example, if $y_i \in \{0, 1\}$,

$$\begin{aligned} P(y_i = 1|y_{-i}) &= \frac{\exp(d_i + d'_{-i}y_{-i} + A'_i y_{-i} + y'_{-i}A_{-i}y_{-i})}{\exp(d_i + d'_{-i}y_{-i} + A'_i y_{-i} + y'_{-i}A_{-i}y_{-i}) + \exp(d'_{-i}y_{-i} + y'_{-i}A_{-i}y_{-i}}) \\ &= \frac{\exp(d_i + A'_i y_{-i})}{\exp(d_i + A'_i y_{-i}) + 1} = \frac{1}{1 + \exp(-d_i - A'_i y_{-i}}) \\ &= \mathrm{sigm}(d_i + A'_i y_{-i}) \end{aligned} \tag{25}$$

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

Two MCMC chains (one for the positive phase and one for the negative phase) are needed for each example 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 80's by the back-propagation algorithm for multi-layer neural network 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 below to train RBMs.

6.3 Restricted Boltzmann Machines

The *Restricted* Boltzmann Machine (RBM) is the building block Deep Belief Networks (DBN) because it shares parametrization with individual layers of a DBN, and because efficient learning algorithms were found to train it. In an RBM, $U = 0$ and $V = 0$ in eq. 21, 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** (Smolensky, 1986), and learning algorithms (beyond the ones for Boltzmann Machines) were discussed in Freund and Haussler (1994). Empirically demonstrated and efficient learning algorithms and variants were proposed more recently (Hinton, 2002; Welling et al., 2005; Carreira-Perpiñan & Hinton, 2005). As a consequence of the lack of input-input and hidden-hidden interactions, the energy function is bilinear,

$$\mathrm{Energy}(x, h) = -b'x - c'h - h'Wx \tag{26}$$

and the factorization of the free energy of the input, introduced with eq. 17 and 19 can be applied with $\beta(x) = b'x$ and $\gamma_i(x, h_i) = h_i W_i x$, where W_i is the row vector corresponding to the i-th row of W. Therefore the free energy of the input (i.e. its unnormalized log-probability) can be computed efficiently:

$$\mathrm{FreeEnergy}(x) = -b'x - \sum_i \log \sum_{h_i} e^{h_i W_i x}. \tag{27}$$

Using the same factorization trick (in eq. 18) due to the affine form of $\mathrm{Energy}(x, h)$ with respect to h, we readily obtain a tractable expression for the conditional probability $P(h|x)$:

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

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

$$P(h_i = 1|x) = \frac{e^{c_i + W_i x}}{1 + e^{c_i + W_i x}} = \text{sigm}(c_i + W_i x). \tag{28}$$

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

$$P(x|h) = \prod_i P(x_i|h) \tag{29}$$

and in the binary case

$$P(x_j = 1|h) = \text{sigm}(b_j + W'_{.j} \cdot h) \tag{30}$$

where $W_{.j}$ is the j-th column of W.

In Hinton et al. (2006) binomial input units are used to encode pixel gray levels 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 Bengio et al. (2007). See Welling et al. (2005) for a general formulation where x and 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 (Freund & Haussler, 1994; Le Roux & Bengio, 2008), 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 bias) can always improve the log-likelihood (Le Roux & Bengio, 2008).

An RBM can also be seen as forming a multi-clustering (see Section 4), as illustrated in Figure 6. Each hidden unit creates a 2-region partition of the input space (with a linear separation). The binary setting of the hidden units identifies one region in input space among all the regions associated with configurations of the hidden units. Note that not all configurations of the hidden units correspond to a non-empty region in input space. This representation is similar to what an ensemble of 2-leaf trees would create.

The sum over an exponential number of configurations can also be seen as a particularly interesting form of mixture, with an exponential number of components (with respect to the number of parameters):

$$P(x) = \sum_h P(x|h)P(h) \tag{31}$$

where $P(x|h)$ is the model associated with the component indexed by configuration h. For example, if $P(x|h)$ is chosen to be Gaussian (see Welling et al. (2005), Bengio et al. (2007)), this is a Gaussian mixture with 2^n components when h has n bits. Of course, these 2^n components cannot be tuned independently because they depend on shared parameters (the RBM parameters). We can see that the Gaussian mean for each component (in the Gaussian case) is obtained as a linear combination $b + W'h$, i.e., each hidden unit bit contributes (or not) a vector W_i in the mean.

6.3.1 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 DBNs are obtained by stacking RBMs, sampling from an RBM enables us to sample from a DBN.

Gibbs sampling in full-blown Boltzmann Machines is slow because one needs to sample both for the positive phase (x clamped to the observed input vector) and for the negative phase (x and h are sampled from the model) and 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 (x, h) can be sampled in two sub-steps in each step of the Gibbs chain. First we sample h given x, and then a new x given h. In general product of experts models, an alternative to Gibbs sampling is hybrid Monte-Carlo, 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 i-th term $\log \sum_{h_i} e^{W_i x h_i}$ in eq. 27 corresponds to an expert, i.e., there is one expert per hidden neuron and one for the input biases. With that special structure, a very efficient Gibbs sampling can be performed. For k Gibbs steps:

$$\begin{aligned} x_0 &\sim \hat{P}(x) \\ h_0 &\sim P(h|x_0) \\ x_1 &\sim P(x|h_0) \\ h_1 &\sim P(h|x_1) \\ &\dots \\ x_k &\sim P(x|h_{k-1}). \end{aligned} \tag{32}$$

Algorithm 1

`RBMupdate`(x_1, ϵ, W, b, c)

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

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)
b is the RBM biases vector for hidden units
c is the RBM biases vector for input units

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i} = 1|x_1)$ (for binomial units, $\mathrm{sigm}(b_i + \sum_j W_{ij} x_{1j})$)
- sample $\mathbf{h}_{1i}$ from $Q(\mathbf{h}_{1i}|x_1)$

end for
for all visible units j **do**
- compute $P(x_{2j} = 1|\mathbf{h}_1)$ (for binomial units, $\mathrm{sigm}(c_j + \sum_i W_{ij} \mathbf{h}_{1i})$)
- sample x_{2j} from $P(x_{2j} = 1|\mathbf{h}_1)$

end for
for all hidden units i **do**
- compute $Q(\mathbf{h}_{2i} = 1|x_2)$ (for binomial units, $\mathrm{sigm}(b_i + \sum_j W_{ij} x_{2j})$)

end for
- $W \leftarrow W + \epsilon(\mathbf{h}_1 x_1' - Q(\mathbf{h}_{2\cdot} = 1|x_2)x_2')$
- $b \leftarrow b + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_{2\cdot} = 1|x_2))$
- $c \leftarrow c + \epsilon(x_1 - x_2)$

6.4 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 (Carreira-Perpiñan & Hinton, 2005). 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.

To obtain this algorithm, the **first approximation** we are going to make is replace the average over all posible inputs (in the second term of eq. 16) 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 updates (which we know to work well (LeCun, Bottou, Orr, & Müller, 1998a)), 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. In any case, we introduce additional variance with this approximation of the gradient.

Running a long MCMC chain is still very expensive. The idea of k-step Contrastive Divergence (CD-k) (Hinton, 1999, 2002) is simple, and involves a **second approximation**, which introduces some bias in the gradient: run the MCMC chain for only k steps *starting from the observed example* x. The CD-k update after seeing example x is therefore

$$\Delta\theta = -\frac{\partial \text{FreeEnergy}(x)}{\partial\theta} + \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial\theta} \tag{33}$$

where $\tilde{x}$ is a sample from our Markov chain 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 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 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 Carreira-Perpiñan and Hinton (2005). In these experiments, taking k larger than 1 gives more precise results, although very good approximations of the solution can obtained even with $k = 1$. Theoretical results (Bengio & Delalleau, 2007) discussed below in Section 7 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 x_1. The stochastic reconstruction x_{k+1} (for CD-k) has a distribution (given x_1) which is in some sense centered around x_1 and becomes less centered around it as k increases, until it becomes the model distribution. The CD-k update will decrease the free energy of the training point x_1 (which would increase its likelihood if all the other free energies were kept constant), and increase the free energy of x_{k+1}, which is in the neighborhood of x_1. Note that x_{k+1} is in the neighborhood of 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 in (LeCun et al., 2006), 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 their neighborhood. 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 model 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 to a good way to identify where that divide should be moved is to compare training examples with samples from the model.

6.5 Model Samples Are Negative Examples

In this section 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 Welling, Zemel, and Hinton (2003), 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 are the training examples 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 $\mathrm{FreeEnergy}(x)$ (or $\mathrm{Energy}(x)$ if we do not introduce explicit hidden variables), given in eq. 16. Now consider a highly regularized two-class probabilistic classifier which is only able to produce an output probability $q(x) = P(y = 1|x)$ barely different from $\frac{1}{2}$ (hopefully on the right side more often than not). Let $q(x) = \mathrm{sigm}(a(x))$, i.e., $a(x)$ is the discriminant function or an unnormalized conditional log-probability, just like the free energy. The average conditional log-likelihood gradient for this probabilistic classifier is

$$\begin{aligned} E_{\hat{P}}\left[\frac{\partial \log P(y|x)}{\partial \theta}\right] &= E_{\hat{P}}\left[\frac{\partial(y \log q(x) + (1-y)\log(1-q(x)))}{\partial \theta}\right] \\ &= E_{\hat{P}}\left[(1-q(x))\frac{\partial a(x)}{\partial \theta}\middle| y=1\right] - E_{\hat{P}}\left[q(x)\frac{\partial a(x)}{\partial \theta}\middle| y=0\right] \\ &\approx \frac{1}{2}E_{\hat{P}}\left[\frac{\partial a(x)}{\partial \theta}\middle| y=1\right] - \frac{1}{2}E_{\hat{P}}\left[\frac{\partial a(x)}{\partial \theta}\middle| y=0\right] \end{aligned} \quad (34)$$

where the last equality is when the classifier is highly regularized: when the output weights are small, $a(x)$ is close to 0 and $q(x) \approx \frac{1}{2}$, so that $(1 - q(x)) \approx q(x)$. 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 (eq. 16), when we interpret training examples as positive examples ($y = 1$) and model samples as negative examples ($y = 0$). 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 (Welling et al., 2003).

6.6 Variants of RBMs

We have already mentioned that it is straightforward to generalize the conditional distributions associated with visible or hidden units, e.g., to any member of the exponential family (Welling et al., 2003). Gaussian units and exponential or truncated exponential units have been proposed or used in Freund and Haussler (1994), Welling et al. (2003), Bengio et al. (2007), Larochelle et al. (2007). 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 h_i and 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.
We review some of the more structural variations that have been proposed on the basic RBM model, in order to increase its expressive power or exploit particular structure in the data.

6.6.1 Lateral Connections

The RBM can be made slightly less restricted by introducing interaction terms or "lateral connections" between visible units. Sampling h from $P(h|x)$ is still easy but sampling x from $P(x|h)$ 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 biases are dependent on the value of h. Osindero and Hinton (2008) propose such a model for capturing image statistics and their results suggest that Deep Belief Nets (DBNs) using such modules generate more realistic image patches than DBNs using 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 work of hidden units for 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 (Olshausen & Field, 1997). The idea proposed in Osindero and Hinton (2008) 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 $P(x|h)$, 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. To reduce sampling variance in CD for this model, Osindero and Hinton (2008) used five damped mean-field steps instead of an ordinary Gibbs chain on the x's: $x_t = \alpha x_{t-1} + (1-\alpha)\text{sigm}(b + Ux_{t-1} + W'h)$, with $\alpha \in (0,1)$.

6.6.2 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 another random variable. For example, consider an RBM for the joint distribution $P(X,H)$ between observed vector X and hidden vector H, with parameters (b,c,W) as per eq. 21, respectively for input biases b, hidden biases c, and the weight matrix W. This idea has been introduced in Taylor, Hinton, and Roweis (2006) for context-dependent RBMs in which the hidden biases c are affine functions of a context variable C. Hence the RBM represents $P(X,H|C)$, or marginalizing over H, $P(X|C)$. In general the parameters θ of the RBM can be written as a parametrized function $\theta = f(C;\omega)$ with parameters ω.
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:

$$\Delta\omega = \Delta\theta \frac{\partial\theta}{\partial\omega}. \tag{35}$$

In the affine case $b = \beta + MC$ (with b, β and C column vectors and M a matrix) studied in Taylor et al. (2006), the CD update on the conditional parameters is simply

$$\begin{aligned} \Delta\beta &= \Delta b \\ \Delta M &= \Delta b\, C' \end{aligned} \tag{36}$$

where the last multiplication is an outer product, and Δb is the update given by CD-k.
This idea has been successfully applied to model conditional distributions $P(x_t|x_{t-1}, x_{t-2}, x_{t-3})$ in sequential data of human motion (Taylor et al., 2006), where 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

$$P(x_1, x_2, \ldots, x_T) \approx P(x_1, \ldots x_k) \prod_{t=k+1}^{T} P(x_t|x_{t-1}, \ldots x_{t-k}). \tag{37}$$

The initial frames can be generated by using special null values as context or using a separate model for $P(x_1, \ldots x_k)$.

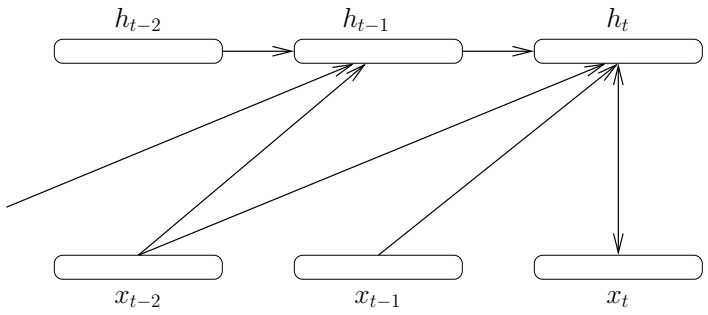

Figure 8: Example of Temporal RBM for modeling sequential data, including dependencies between the hidden variables. The double-arrow arc indicates an undirected connection, i.e. an RBM. The single-arrow arcs indicate conditional dependency: the (x_t, h_t) RBM is conditionned by the values of the past inputs and past hidden vectors.

As demonstrated in Memisevic and Hinton (2007), it can be useful to make not just the biases 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 x_i, a hidden unit h_j, and a context unit c_k through interaction parameters U_{ijk}. This approach has been used with X an image and C the previous image in a video, and the model learns to capture *flow fields* (Memisevic & Hinton, 2007).
Probabilistic models of sequential data with hidden variables H_t (called **state**) can gain a lot by capturing the temporal dependencies between the hidden variable at different times t in the sequence. This is what allows **Hidden Markov Models** (HMMs) (Rabiner & Juang, 1986) to capture dependencies in a long sequence even if the model only considers the hidden variable to be a Markov chain of order 1 (where the direct dependence is only between H_t and H_{t+1}). Whereas the hidden variable representation H_t in HMMs is *local* (all the possible values of H_t are enumerated and specific parameters associated with each of these values), **Temporal RBMs** have been proposed (Sutskever & Hinton, 2007) 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

$$P(H_t, X_t | H_{t-1}, X_{t-1}, \ldots, H_{t-k}, X_{t-k}) \tag{38}$$

where the context is $C_t = (H_{t-1}, X_{t-1}, \ldots, H_{t-k}, X_{t-k})$, as illustrated in Figure 8. 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 not anymore tractable. Instead, Sutskever and Hinton (2007) propose to use a mean-field filtering approximation of the hidden sequence posterior.

6.6.3 Factored RBMs

In several probabilistic language models, it has been proposed to learn a distributed representation of each word (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990; Miikkulainen & Dyer, 1991; Bengio et al., 2001, 2003). For an RBM that models a sequence of words, it would be convenient to have a parametrization that automatically learns a distributed representation for each word in the vocabulary. This is essentially what Mnih and Hinton (2007) propose. They use a factorization of the weight matrix W into two factors, one that depends on the location in the input subsequence, and one that does not. Consider the computation of the hidden units' probabilities given the input subsequence $(w_1, w_2, \ldots, w_k)$, where each word w_t is represented by a one-hot vector v_t (all 0's except for a 1 at position w_t) and these vectors are concatenated into the input vector $x = (v_1, \ldots, v_k)$. Instead of applying directly a matrix W to x, do the following. First, each word symbol w_t is mapped through a matrix R to a d-dimensional vector $R_{.,w_t} = Rv_t$,

for $t \in \{1 \dots k\}$; second, the concatenated vectors $(R_{.,w_1}, R_{.,w_2}, \dots, R_{.,w_k}) = (Rv_1, \dots Rv_k)$ are multiplied by a matrix B. Hence $W = B(R\ R\ \dots R)$, where $(R\ R\ \dots R)$ indicates concatenation (not product) of R. This model has produced better out-of-sample log-likelihood than state-of-the-art language models based on n-grams (Mnih & Hinton, 2007). This factorization can be combined with the temporal RBM idea introduced above, yielding further improvements in generalization performance (Mnih & Hinton, 2007).

7 Truncations of the Log-Likelihood 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 autoassociators (eq. 5). The inspiration for this derivation comes from Hinton et al. (2006): first from the idea (explained in Section 11.1) that the Gibbs chain can be associated with an infinite directed graphical model (which here we associate to an expansion of the log-likelihood and of its gradient), and second that the convergence of the chain justifies Contrastive Divergence (since the expected value of eq. 33 becomes equivalent to eq. 15 when the chain sample $\tilde{x}$ comes from the model).
Consider a converging Markov chain $x_t \Rightarrow h_t \Rightarrow x_{t+1} \Rightarrow \dots$ defined by conditional distributions $P(h_t|x_t)$ and $P(x_{t+1}|h_t)$. A sufficient condition for convergence is that it mixes, i.e., one can reach any state from any state in finite time.
The following Lemma, demonstrated in (Bengio & Delalleau, 2007), shows that the by consecutive application of Bayes rule, one can expand the log-likelihood in a series that involves the samples in the Gibbs chain.

Lemma 7.1. *Consider the Gibbs chain $x_1 \Rightarrow h_1 \Rightarrow x_2 \Rightarrow h_2 \dots$ starting at data point x_1. The log-likelihood can be expanded as follows for any path of the chain:*

$$\log P(x_1) = \log P(x_t) + \sum_{s=1}^{t-1} \log \frac{P(x_s|h_s)}{P(h_s|x_s)} + \log \frac{P(h_s|x_{s+1})}{P(x_{s+1}|h_s)} \tag{39}$$

and consequently, since this is true for any path:

$$\log P(x_1) = E[\log P(x_t)] + \sum_{s=1}^{t-1} E\left[\log \frac{P(x_s|h_s)}{P(h_s|x_s)} + \log \frac{P(h_s|x_{s+1})}{P(x_{s+1}|h_s)}\right] \tag{40}$$

where the expectation is over the Markov chain, conditional on x_1.

In the limit $t \to \infty$, the last term is just the entropy of distribution $P(x)$. Note that the terms do not vanish as $t \to \infty$, so as such this expansion does not justify truncating the series to approximate the log-likelihood. We will see that reconstruction error, often used for monitoring training progress of RBMs, is closely related to the first term in the series.
Now consider the corresponding gradient series. To prove the theorem, the following simple lemma, which we use later, is very useful:

Lemma 7.2. *For any model $P(Y)$ with parameters θ,*

$$E\left[\frac{\partial \log P(Y)}{\partial \theta}\right] = 0 \tag{41}$$

when the expected value is taken according to $P(Y)$.

Proof. We start from the sum to 1 constraint on $P(Y)$, differentiate and obtain the Lemma. To obtain the last line below we use the fact that for any function $f(\theta)$, we have $\frac{\partial f(\theta)}{\partial \theta} = f(\theta)\frac{\partial \log f(\theta)}{\partial \theta}$.

$$\begin{aligned} E[1] &= \sum_y P(Y=y) = 1 \\ \frac{\partial \sum_y P(Y=y)}{\partial \theta} &= \frac{\partial 1}{\partial \theta} = 0 \\ \sum_y P(Y=y)\frac{\partial \log P(Y=y)}{\partial \theta} &= 0 \end{aligned}$$

□

The following theorem can then be proved (Bengio & Delalleau, 2007).

Theorem 7.3. *Consider the converging Gibbs chain $x_1 \Rightarrow h_1 \Rightarrow x_2 \Rightarrow h_2 \ldots$ starting at data point x_1. The log-likelihood gradient can be expanded in a converging series as follows, where all expectations are conditional on x_1:*

$$\begin{aligned} \frac{\partial \log P(x_1)}{\partial \theta} &= \sum_{s=1}^{t-1} \left(E\left[\frac{\partial \log P(x_s|h_s)}{\partial \theta}\right] + E\left[\frac{\partial \log P(h_s|x_{s+1})}{\partial \theta}\right]\right) \\ &+ E\left[\frac{\partial \log P(x_t)}{\partial \theta}\right] \end{aligned} \tag{42}$$

with the terms in s converging to 0 as $s \to \infty$, and the final term (in t) also converges to 0, as $t \to \infty$.

Since the k-th term becomes small as k increases, that justifies truncating the chain to k steps. Note how the sums in the above expansion can be readily replaced by easy to obtain samples (for the first k steps in the Gibbs chain). This gives rise to a stochastic gradient, whose expected value is the exact expression associated with a truncation of the above log-likelihood gradient expansion. Finally, it can be shown (Bengio & Delalleau, 2007) that truncating to the first k steps gives a parameter update that is exactly the CD-k update in the case of a binomial RBM.

Corollary 7.4. *When considering only the terms arising of the first k steps in the Gibbs chain $x_1 \Rightarrow h_1 \Rightarrow x_2 \Rightarrow h_2 \Rightarrow \ldots x_k \Rightarrow h_k$, the unbiased stochastic estimator of the gradient of the truncated log-likelihood expansion of theorem 7.3 (with expectations replaced by samples in the chain) equals the CD-k update in the case of a binomial RBM.*

Experiments and theory support the idea that CD-k yields better and faster convergence (in terms of number of iterations) than CD-$(k-1)$ (but the computational overhead might not always be worth it). This is because smaller k corresponds to more bias in the estimation of the log-likelihood gradient. So CD-1 corresponds to taking the first two terms in the expansion (one sample of $h_1|x_1$ and one sample of $x_2|h_1$). What about taking only the first one? The first term in the log-likelihood gradient expansion is

$$\sum_{h_1} P(h_1|x_1)\frac{\partial \log P(x_1|h_1)}{\partial \theta} \tag{43}$$

Now consider a mean-field approximation of the above, in which instead of the average over all h_1 configurations according to $P(h_1|x_1)$ one replaces h_1 by its average configuration $\hat{h}_1 = E[h_1|x1]$, yielding:

$$\frac{\partial \log P(x_1|\hat{h}_1)}{\partial \theta} \tag{44}$$

which is minus the gradient of **reconstruction error**,

$$-\log P(x_1|\hat{h}_1) \tag{45}$$

typically used to train autoassociators.
So we have found that the truncation of the log-likelihood expansion gives rise to first approximation (1 term) to roughly reconstruction error (through a biased mean-field approximation), with slightly better approximation (2 terms) to CD-1 (approximating the expectation by a sample), and with more terms to CD-k. Note that reconstruction error is deterministically computed and for this reason has been used to track progress when training RBMs with CD. Since reconstruction error and CD-k are complementary in terms of bias and variance (as estimators of the log-likelihood gradient), it might be interesting to explore combinations of them: a low-variance high-bias estimator (reconstruction error gradient) might be more useful at the beginning of training (where having a precise estimation of the gradient is less important) whereas the low-bias high-variance estimator (CD-k) would be more useful to achieve training convergence.

8 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 (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 x and hidden vector h whose energy function is such that $P(h|x)$ and $P(x|h)$ both factorize. This definition can be formalized in terms of the parametrization of the energy function:

Proposition 8.1. *The energy function associated with a model of the form of eq. 11 such that* $P(h|x) = \prod_i P(h_i|x)$ *and* $P(x|h) = \prod_j P(x_j|h)$ *must have the form*

$$\text{Energy}(x, h) = \sum_j \phi_j(x_j) + \sum_i \xi_i(h_i) + \sum_{i,j} \eta_{i,j}(h_i, x_j). \tag{46}$$

Proof. To achieve factorization of $P(h|x)$ we have already shown that the energy function must be writable as a sum over i (with one term per h_i), in eq. 18. This gives us the constraint that $\text{Energy}(x, h)$ can be written as $\text{Energy}(x, h) = -\beta(x) + \sum_i \gamma_i(x, h_i)$, for some β and γ_i. Using the same arguments but inverting the roles of x and h, we obtain that the constraint $\text{Energy}(x, h) = -\alpha(h) + \sum_j \rho_j(x_j, h)$ for some α and ρ_j. Clearly if $\text{Energy}(x, h)$ can be written as in eq. 46, then these two constraints are satisfied. On the other hand, consider adding a term of a different from (not depending only on h_i, only on x_j, or only on a pair (h_i, x_j) to the right hand side of eq. 46. Then one of the above two constraints would be violated. Therefore the above equation is the most general formulation that satisfies both factorization assumptions. □

In the case where the hidden and input values are binary, this new formulation does not actually bring any additional power of representation. Indeed, $\eta_{i,j}(h_i, x_j)$, which can take at most four different values according to the 2×2 configurations of (h_i, x_j) could always be rewritten as a second order polynomial in (h_i, x_j): $a + bh_i + cx_j + dh_i x_j$. However, b and c can be folded into the bias terms and a into a global addititive constant which does not matter (because it gets cancelled by the partition function).
On the other hand, when x or h are real-valued, one could imagine higher-capacity modeling of the (h_i, 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 $P(x_j|h)$ or $P(h_i|x)$ would be tractable even if the $\eta_{i,j}$ are complicated functions, simply because these are 1-dimensional densities from which efficient approximate sampling and numerical integration are easy (e.g., by computing cumulative sums of the density over nested subintervals 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 h are hidden and that we can have as many hidden

units as we want that we still have full expressive power over possible marginal distributions in x. Other variants of RBMs discussed in Section 6.6 allow to introduce three-way interactions (Memisevic & Hinton, 2007).
Can Contrastive Divergence be applied to this generalized RBM formulation? Clearly, theorem 7.3 can still be applied. Furthermore, it can be shown, generalizing corollary 7.4 that considering only the first k steps of the Gibbs chain in the log-likelihood gradient expansion, one obtains an update rule similar to CD-k for binomial RBMs.

Proposition 8.2. *Consider a generalized RBM, with the energy function as in eq. 46. When considering only the terms arising of the first k steps in the Gibbs chain $x_1 \Rightarrow h_1 \Rightarrow x_2 \Rightarrow h_2 \Rightarrow \dots x_k \Rightarrow h_k$, with the unbiased stochastic estimator of the gradient of the truncated log-likelihood expansion of theorem 7.3, all the intermediate gradient terms cancel each other and the gradient estimator only depends directly on the first pair (x_1, h_1) and on the last pair (x_k, h_k), e.g., for θ a parameter of $\eta_{i,j}$:*

$$\sum_{s=1}^{t-1} E\left[\frac{\partial \log P(x_s|h_s)}{\partial \theta} + \frac{\partial \log P(h_s|x_{s+1})}{\partial \theta}\right] = E\left[\sum_{i,j} \frac{\partial \eta_{i,j}(h_{1,i}, x_{1,j})}{\partial \theta} - \sum_{i,j} \frac{\partial \eta_{i,j}(h_{k,i}, x_{k,j})}{\partial \theta}\right] \quad (47)$$

where $h_{k,i}$ is the i-th element of the k-th hidden vector h_k in the chain, and similarly for $x_{k,j}$, and the expectation is over the Markov chain, conditionned on x_1.

Proof. Note that the ϕ_j and ξ_i terms can be represented by extra $\eta_{i,j}$ terms so we will ignore them in the proof. By definition of our energy function and using the factorization of the conditionals shown above, we have

$$P(h_{s,i}|x_s) = \frac{\exp(\sum_j \eta_{i,j}(h_{s,i}, x_{s,j}))}{\exp(\sum_{\tilde{h}_{s,i}} \sum_j \eta_{i,j}(\tilde{h}_{s,i}, x_{s,j}))} \quad (48)$$

and

$$P(x_{s+1,j}|h_s) = \frac{\exp(\sum_i \eta_{i,j}(h_{s,i}, x_{s+1,j}))}{\exp(\sum_{\tilde{x}_{s+1,j}} \sum_i \eta_{i,j}(h_{s,i}, \tilde{x}_{s+1,j}))} \quad (49)$$

Differentiating them and taking expectations with respect to the Markov chain, we find that the gradient of the denominator of $\log P(x_s|h_s)$ cancels the gradient of the numerator of $\log P(h_s|x_{s+1})$, and similarly that the gradient of the denominator of $\log P(h_s|x_{s+1})$ cancels the gradient of the numerator of $\log P(x_{s+1}|h_{s+1})$. Hence, ignoring the remainder $E\left[\frac{\partial \log P(x_{k+1})}{\partial \theta}\right]$ due to truncation of the series, there only remains from eq. 42 the gradient of the numerator of $\log P(x_1|h_1)$ and the gradient of the denominator of $\log P(h_k|x_{k+1})$. □

Therefore, when generalizing RBMs with an energy function of the form of eq. 46, a Gibbs chain can still be run easily (thanks to Proposition 8.1), either to sample data from the model or for learning, and a CD-k algorithm can be run to gradually tune the parameters, with the parameter update given by

$$\Delta\theta = \sum_{i,j} \frac{\partial \eta_{i,j}(h_{1,i}, x_{1,j})}{\partial \theta} - \sum_{i,j} \frac{\partial \eta_{i,j}(h_{k,i}, x_{k,j})}{\partial \theta} \quad (50)$$

with ϵ a learning rate for the stochastic gradient descent. Note that in most parametrizations we would have a particular element of θ only depend on a particular $\eta_{i,j}$ (and no sum is needed). We recover Algorithm 1 when $\eta_{i,j}(h_{1,i}, x_{1,j}) = W_{ij} h_{1,i} x_{1,j}$ and the other variants described in (Welling et al., 2005; Bengio et al., 2007) for different forms of the energy and allowed set of values for hidden and input units.

9 Stacked Autoassociators

Autoassociators have been used as building blocks to build a deep multi-layer neural network (Bengio et al., 2007; Ranzato et al., 2007; Larochelle et al., 2007). The training procedure is simpler than with Deep Belief Networks, so we start with it, noting that many variations on that scheme are possible:

1. Train the first layer as an autoassociator to minimize some form of reconstruction error of the raw input. This is purely unsupervised.
2. The hidden units' outputs in the autoassociator are now used as input for another layer, also trained to be an autoassociator. Again, we only need unlabeled examples.
3. Iterate as in (2) to add the desired number of layers.
4. Take the last hidden layer output as input to a supervised layer and initialize its parameters (either randomly or by supervised training, keep 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 autoassociators into a very deep autoassociator and fine-tune the global reconstruction error, as in (Hinton & Salakhutdinov, 2006).

The hope is that the unsupervised initialization in a greedy layer-wise fashion has put the parameters of all the layers in a region of parameter space from which a good local optimum can be reached by local descent. This indeed appears to happen in a number of tasks (Bengio et al., 2007; Ranzato et al., 2007; Larochelle et al., 2007).

The principle is exactly the same as the one previously proposed for training Deep Belief Networks (Hinton et al., 2006), but using autoassociators instead of RBMs. Comparative experimental results in (Bengio et al., 2007; Larochelle et al., 2007) suggest that Deep Belief Networks typically (but not systematically) have a slight edge over stacked autoassociators, maybe 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.

An advantage of using autoassociators instead of RBMs as the unsupervised building block of a deep architecture is that almost any parametrizations of the layers are 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 autoassociators 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.

Note that the above algorithm can be naturally in the semi-supervised setting, where only a fraction of the training examples are associated with a supervision label. For unlabeled examples only an unsupervised criterion is used (e.g., reconstruction error at each level or over the whole network), whereas for supervised examples the supervised criterion is used. For labeled examples, both criteria can be combined. Combining both criteria has been found useful not only at the fine-tuning stage (where all the layers are jointly optimized) but also during the greedy layerwise stage (Bengio et al., 2007). This form of *partial supervision* has been found useful in cases where the true input distribution is not very informative of the target conditional distribution that one wants to capture for the supervised task.

10 Deep Belief Networks

A Deep Belief Network (Hinton et al., 2006) with ℓ layers models the joint distribution between observed vector x and ℓ hidden layers $\mathbf{h}^k$ as follows:

$$P(x, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = \left(\prod_{k=1}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell) \tag{51}$$

where $x = \mathbf{h}^0$, $P(\mathbf{h}^{k-1}|\mathbf{h}^k)$ is a conditional distribution for visible hidden units in an RBM associated with level k of the DBN, and $P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$ is the visible-hidden joint distribution in the top-level RBM. This is illustrated in Figure 9.

Algorithm 2

`TrainUnsupervisedDBN`($\widehat{p}, \epsilon, L$,n,W,b)
Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM by contrastive divergence.
$\widehat{p}$ is the input training distribution for the network
ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence
L is the number of layers to train
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L

- initialize $b^0 = 0$

for $\ell = 1$ to L **do**
- initialize $W^i = 0$, $b^i = 0$

while not stopping criterion **do**
- sample $\mathbf{h}^0 = x$ from $\widehat{p}$

for $k = 1$ to $\ell - 1$ **do**
- sample $\mathbf{h}^k$ from $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$

end for
- `RBMupdate`($\mathbf{h}^{\ell-1}, \epsilon, W^\ell, b^\ell, b^{\ell-1}$) {thus providing $Q(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$ for future use}

end while
end for

When we train the DBN in a greedy layerwise fashion, as illustrated with the pseudo-code of Algorithm 2, each layer is initialized as an RBM, and we denote $Q(\mathbf{h}^k, \mathbf{h}^{k-1})$ the k-th RBM trained in this way. 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 x. To obtain an approximate posterior or representation for all the levels, we use the following procedure. First sample $\mathbf{h}^1 \sim Q(\mathbf{h}^1|x)$ from the first-level RBM, or alternatively with a mean-field approach use $\hat{\mathbf{h}}^1 = E[\mathbf{h}^1|\mathbf{h}^0]$ instead of a sample of $\mathbf{h}^1$, where the expectation is over the RBM distribution $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$. This is just the output probabilities of the hidden units, in the common case where they are binomial units: $\hat{\mathbf{h}}^1_i = \mathrm{sigm}(b^1 + W^1_i x)$. Taking either the sample $\mathbf{h}^1$ or the mean-field vector $\hat{h}^1$ as input for the second-level RBM, compute $\hat{h}^2$ or a sample $\mathbf{h}^2$, etc. until the last layer. A sample of the DBN generative model for 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 $\mathbf{h}^\ell \sim P(h^\ell|h^{\ell-1})$ and $\mathbf{h}^{\ell-1} \sim P(h^{\ell-1}|h^\ell)$, as outlined in Section 6.3.1. 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.

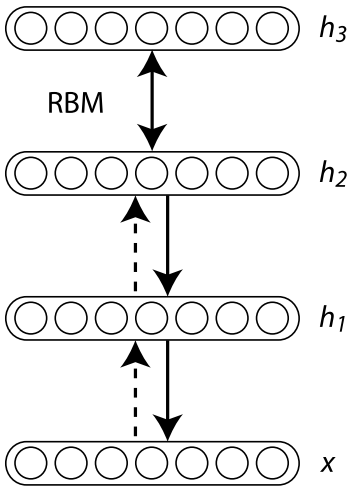

Figure 9: Deep Belief Network as a generative model (generative path, with bold arcs) and a means to extract multiple levels of representation of the input (recognition path, with dashed arcs). The top two layers $\mathbf{h}^2$ and $\mathbf{h}^3$ form an RBM (for their joint distribution). The lower layers form a directed graphical model (sigmoid belief net $\mathbf{h}^2 \Rightarrow \mathbf{h}^1 \Rightarrow x$) and the prior for the penultimate layer $\mathbf{h}^2$ is provided by the top-level RBM.

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. $x = \mathbf{h}^0$ is the DBN sample.

The principle of greedy layer-wise unsupervised training of each layer on top of the previously trained ones can be applied with RBMs as the building blocks for each layer (Hinton et al., 2006; Hinton & Salakhutdinov, 2006; Bengio et al., 2007; Salakhutdinov & Hinton, 2007).

1. Train the first layer as an RBM that models the raw input $x = \mathbf{h}^0$ as its visible layer.

2. As outlined above, use that first layer to obtain a representation of the input data that will be used as data for the second layer. Two common solutions are to take hidden layer samples of $\mathbf{h}^1|\mathbf{h}^0$ or the real values $\hat{h}^1 = E(\mathbf{h}^1|\mathbf{h}^0)$) for this representation.

3. Train the second layer as an RBM, taking the transformed data ($\mathbf{h}^1|x$ or $\hat{h}^1(x)$) as training example (for the visible layer of that RBM).

4. Iterate (2 and 3) for the desired number of layers, each time propagating upward either samples or mean values.

5. Fine-tune all the parameters of this deep architecture with respect to a proxy for the DBN log-likelihood, or with respect to a supervised training criterion (after adding extra learning machinery to convert the learned representation into supervised predictions).

The remark made at the end of Section 9 about semi-supervised and partially supervised training also applies to DBNs. Combining labeled data and unlabeled data is simple with DBNs, and the partially supervised setting has been found experimentally useful for some tasks (Bengio et al., 2007).

11 Stochastic Variational Bounds for Joint Optimization of DBN Layers

In this section we discuss mathematical underpinnings of training algorithms for DBNs. 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 (Hinton et al., 2006) and described in Section 10. Starting from eq. 51 for a DBN joint distribution, writing h for $\mathbf{h}^1$ (the first level hidden vector) to lighten notation, and introducing an arbitrary conditional distribution $Q(h|x)$ we have

$$\begin{aligned}
\log P(x) &= \log \sum_h P(x,h) \\
&= \log \sum_h \frac{Q(h|x)P(x,h)}{Q(h|x)} \\
&\geq \sum_h Q(h|x) \log \frac{P(x,h)}{Q(h|x)} \\
&= \sum_h Q(h|x) \left(\log P(x,h) - \log Q(h|x)\right) \\
&= H_{Q(h|x)} + \sum_h Q(h|x) \left(\log P(h) + \log P(x|h)\right). \qquad (52)
\end{aligned}$$

where $H_{Q(h|x)}$ is the entropy of the distribution $Q(h|x)$. To see what the inequality is missing out, we can use another derivation, which is again true for any $Q(h|x)$ and P. First multiply by $1 = \sum_h Q(h|x)$, then use $P(x) = \frac{P(x,h)}{P(h|x)}$, and multiply by $1 = \frac{Q(h|x)}{Q(h|x)}$ and expand the terms:

$$\begin{aligned}
\log P(x) &= (\sum_h Q(h|x)) \log P(x) = \sum_h Q(h|x) \log \frac{P(x,h)}{P(h|x)} \\
&= \sum_h Q(h|x) \log \frac{P(x,h)}{P(h|x)} \frac{Q(h|x)}{Q(h|x)} \\
&= H_{Q(h|x)} + \sum_h Q(h|x) \log P(x,h) + \sum_h Q(h|x) \log \frac{Q(h|x)}{P(h|x)} \\
&= KL(Q(h|x)||P(h|x)) + H_{Q(h|x)} + \sum_h Q(h|x) \left(\log P(h) + \log P(x|h)\right). \qquad (53)
\end{aligned}$$

So the missing term in inequality 52 is the Kullback-Liebler divergence between the two conditional distributions $Q(h|x)$ and $P(h|x)$. Whereas we have chosen to use P to denote probabilities under the DBN, let us use Q to denote probabilities under an RBM (which we will call the first level RBM) , and in the equations choose $Q(h|x)$ to be the hidden-given-visible conditional distribution of that first level RBM. We define that first level RBM such that $Q(x|h) = P(x|h)$. In general $P(h|x) \neq Q(h|x)$. This is because although the marginal $P(h)$ on the first layer hidden vector $\mathbf{h}^1 = h$ is determined by the upper layers in the DBN, the RBM marginal $Q(h)$ only depends on the parameters of the RBM.

11.1 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 $\mathbf{h}^1$ marginal, i.e. $P(\mathbf{h}^1) = Q(\mathbf{h}^1)$, as long the dimension of $\mathbf{h}^2$ equals the dimension of $\mathbf{h}^0 = x$.

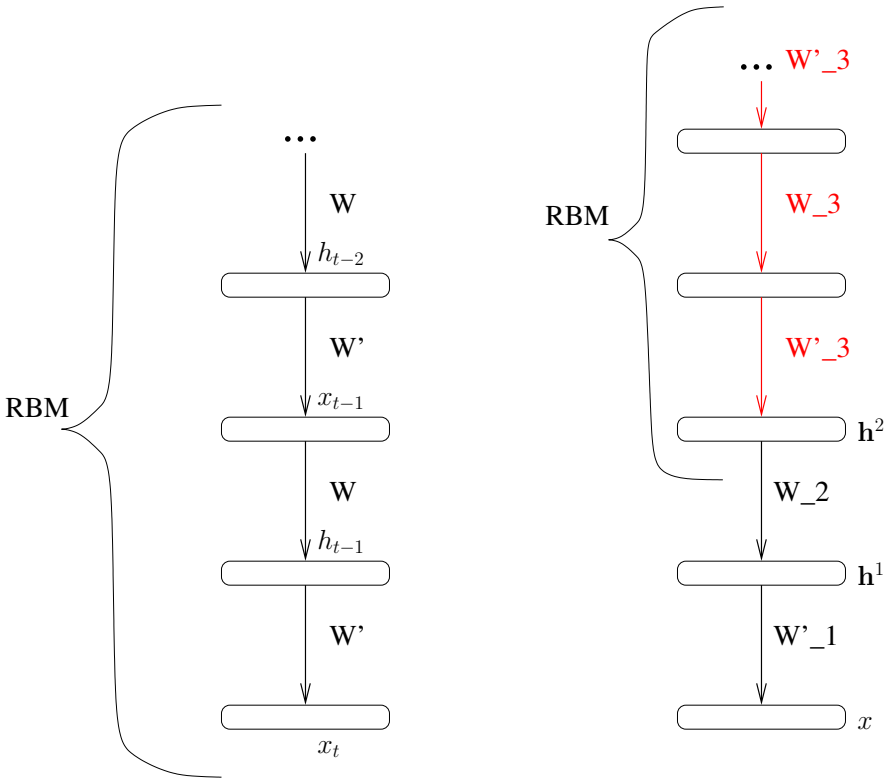

Figure 10: An RBM can be unfolded as an infinite directed belief network with tied weights (see text), left. The weight matrix W or its transpose are used depending on the parity of the layer index. This sequence of random variables corresponds to a Gibbs Markov chain to generate x_t (for t large). On the 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).

To see this, consider a second 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 explanation is given in (Hinton et al., 2006): consider the infinite Gibbs sampling Markov chain starting at $t = -\infty$ and terminating at $t = 0$, alternating between 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 10. This argument also shows that a 2-layer DBN in which the second level has weights equal to the transpose of the first level weights is equivalent to a single RBM.

11.2 Variational Justification of Greedy Layerwise Training

Here we discuss the argument made in Hinton et al. (2006) that adding one RBM layer improves the likelihood of a DBN. Let us suppose we have trained an RBM to model x, which provides us with a model $Q(x)$ expressed through two conditionals $Q(\mathbf{h}^1|x)$ and $Q(x|\mathbf{h}^1)$. Exploiting the argument in the previ-

ous subsection, let us now initialize an equivalent 2-layer DBN, i.e., generating $P(x) = Q(x)$, by taking $P(x|\mathbf{h}^1) = Q(x|\mathbf{h}^1)$ 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 eq. 53 above, and the objective of improving the DBN likelihood by changing $P(\mathbf{h}^1)$, i.e., keeping $P(x|\mathbf{h}^1)$ and $Q(\mathbf{h}^1|x)$ fixed but allowing the second level RBM to change. Starting from $P(x|\mathbf{h}^1) = Q(x|\mathbf{h}^1)$, the KL term is zero and the entropy term in eq. 53 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 $\log P(x)$. We are also guaranteed that further improvements of the $P(\mathbf{h}^1)$ term (i.e. further training of the second RBM) 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, as argued in Hinton et al. (2006). This justifies training the second RBM to maximize the expectation over the training set of $\sum_{\mathbf{h}^1} Q(\mathbf{h}^1|x) \log P(\mathbf{h}^1)$.
The second-level RBM is thus trained to maximize

$$\sum_{x, \mathbf{h}^1} \hat{P}(x) Q(\mathbf{h}^1|x) \log P(\mathbf{h}^1) \tag{54}$$

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 $\hat{P}(x)Q(\mathbf{h}^1|x)$. If there was no constraint on $P(\mathbf{h}^1)$, the maximizer of the above training criterion would be its "empirical" or target distribution

$$P^*(\mathbf{h}^1) = \sum_x \hat{P}(x) Q(\mathbf{h}^1|x). \tag{55}$$

If we keep the first-level RBM fixed, then the second-level RBM could therefore be trained as follows: sample x from the training set, then sample $\mathbf{h}^1 \sim Q(\mathbf{h}^1|x)$, and consider that h as a training sample for the second-level RBM.
The same argument can be made to justify adding a third layer, etc. We obtain the greedy layer-wise training procedure outlined in Section 10. 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 (Hinton et al., 2006; Bengio et al., 2007), 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 second RBM (and this includes adding extra layers), there is no guarantee that $\log P(x)$ (in average over the training set) will monotonically increase. As our lower bound continuous to increase, the actual log-likelihood could start decreasing. Let us examine more closely how this could happen. It would require the KL 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 $P(\mathbf{h}^1|x)$ (from the DBN) and $Q(\mathbf{h}^1|x)$ (from the RBM) deviate more and more (since $P(\mathbf{h}^1|x) \propto P(x|\mathbf{h}^1)P(\mathbf{h}^1)$), making the KL term in eq. 53 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.
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 $\hat{P}(x)$. This is because $P^*(\mathbf{h}^1)$ was obtained by applying one sub-step of an RBM Gibbs chain on examples from $\hat{P}(x)$, and we know that applying many Gibbs steps would yield data from that RBM.
Unfortunately, when we train 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. Le Roux and Bengio (2008) have proposed considering alternatives to Contrastive Divergence for training RBMs destined to initialize intermediate layers of a DBN. The idea is to consider that $P(h)$ 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 $P(h)$ will be: it is simply the stochastic transformation of the empirical distribution through the stochastic mapping $Q(h|x)$ of the first RBM (or previous RBMs). Plugging this back into the expression for $\log P(x)$, 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 (Le Roux & Bengio, 2008) 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 h. Tractable approximations of it might be considered. Another interesting alternative, explored in the next section, is to directly work on joint optimization of all the layers of a DBN.

11.3 Joint Unsupervised Training of All the Layers

We discuss here how one could train the whole DBN with respect to the unsupervised log-likelihood. The log-likelihood decomposition in eq. 53

$$\log P(x) = KL(Q(h|x)||P(h|x)) + H_{Q(h|x)} + \sum_h Q(h|x) \left(\log P(h) + \log P(x|h)\right). \tag{56}$$

can be used not only to justify the greedy training algorithm, but also to justify learning algorithms in which all the layers of a DBN are simultaneously updated, maybe after a greedy layerwise initialization phase.
The top level of the DBN would be trained as an RBM, i.e., choosing $P(h)$ to maximize $\sum_h Q(h|x) \log P(h)$, where h is the penultimate layer of the DBN, and the top level RBM represents the joint distribution between the penultimate and top layer of the DBN.
Instead of keeping the lower levels fixed, if we want to improve them while taking into account the particulars of higher levels, we can return to eq. 53 and compute an estimate of the gradient of the log-likelihood with respect to $P(x|h)$ and $Q(h|x)$. To simplify the exposition we only consider the case of a 2-level DBN, but the same principle can be easily generalized to any number of levels.
The gradient of the entropy of $Q(h|x)$ is easy to estimate stochastically, using one or more samples of $h \sim Q(h|x)$. Consider a parameter θ that influences $Q(h|x)$. Using first $\frac{\partial y}{\partial x} = y \frac{\partial \log y}{\partial x}$, and Lemma 7.2 in the second line,

$$\begin{aligned} \frac{\partial H_{Q(h|x)}}{\partial \theta} &= -\sum_h Q(h|x) \frac{\partial \log Q(h|x)}{\partial \theta} - \sum_h Q(h|x) \log Q(h|x) \frac{\partial \log Q(h|x)}{\partial \theta} \\ &= -\sum_h Q(h|x) \log Q(h|x) \frac{\partial \log Q(h|x)}{\partial \theta}. \end{aligned} \tag{57}$$

A stochastic gradient with respect to a parameter θ of the first level that influences $P(x|h)$ is also easy to obtain, using a similar derivation:

$$\frac{\partial \sum_h Q(h|x) \log P(x|h)}{\partial \theta} = \sum_h Q(h|x) \frac{\partial \log P(h|x)}{\partial \theta} + \sum_h Q(h|x) \log P(h|x) \frac{\partial \log Q(h|x)}{\partial \theta} \tag{58}$$

In both cases, we sample x from the training set and $h \sim Q(h|x)$ and use the gradients $\log Q(h|x) \frac{\partial \log Q(h|x)}{\partial \theta}$ and $\frac{\partial \log P(h|x)}{\partial \theta} + \log P(h|x) \frac{\partial \log Q(h|x)}{\partial \theta}$. Note that these estimators could have high variance because $\log Q(h|x)$ and $\log P(h|x)$ could be arbitrarily large. In fact their variance might grow linearly with the dimension of the hidden vector.
The gradient of the KL divergence is more problematic, because we do not have a simple expression for $P(h|x)$. The KL term in eq. 53 could potentially be ignored since it is positive and we would be optimizing a lower bound on the log-likelihood. Instead, an approximation has been used in the **wake-sleep algorithm** for sigmoid belief networks (Hinton et al., 1995). The idea is to minimize the other KL divergence,

$KL(P(h|x)||Q(h|x))$. Indeed if we sample from the DBN, we obtain an (h, x) tuple from $P(x, h)$ that can be used as target for $Q(h|x)$. Again using Lemma 7.2 and $\frac{\partial y}{\partial x} = y\frac{\partial \log y}{\partial x}$,

$$\begin{aligned}\frac{\partial KL(P(h|x)||Q(h|x))}{\partial \theta} &= \sum_h P(h|x)\left(\log \frac{P(h|x)}{Q(h|x)}\frac{\partial \log P(h|x)}{\partial \theta} + \frac{\partial \log P(h|x)}{\partial \theta} - \frac{\partial \log Q(h|x)}{\partial \theta}\right) \\ &= \sum_h P(h|x)\left(\log \frac{P(h|x)}{Q(h|x)}\frac{\partial \log P(h|x)}{\partial \theta} - \frac{\partial \log Q(h|x)}{\partial \theta}\right). \quad (59)\end{aligned}$$

As before, we can obtain a stochastic estimator, but the $\log \frac{P(h|x)}{Q(h|x)}\frac{\partial \log P(h|x)}{\partial \theta}$ term might have high variance. In the wake-sleep algorithm (Hinton et al., 1995) and its contrastive version for DBNs (Hinton et al., 2006), the parameters of $Q(h|x)$ and the parameters of $P(x|h)$ are decoupled. In the context of a DBN, for all levels except the top one, there is no reason to believe that the optimal "generative weights" (those used in $P(x|h)$) have to be equal (transposed) to the "recognition weights" (those used in $Q(h|x)$). The wake-sleep algorithm provides an update rule for both. We know that the true posterior $P(h|x)$ does not necessarily factorize (cannot be written as $\prod_i P(h_i|x)$) whereas $Q(h|x)$ does factorize. The algorithm proceeds in two phases: the wake phase and the sleep phase. In the **wake phase**, we start from a training sample x and compute samples from the approximate posteriors given by the $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$'s at each level (starting from $\mathbf{h}^0 = x$). These samples $(\mathbf{h}^0, \mathbf{h}^1, \ldots \mathbf{h}^{\ell-1})$ provide fully observed training data for updating the $P(\mathbf{h}^{k-1}|\mathbf{h}^k)$ generative distributions, i.e, a stochastic step in the direction of the following gradient is performed

$$\sum_{\mathbf{h}^1,\ldots,\mathbf{h}^{\ell-1}} Q(\mathbf{h}^{\ell-1}|\mathbf{h}^{\ell-2}) \ldots Q(\mathbf{h}^1|x) \prod_{k=1}^{\ell-1} \frac{\partial \log P(\mathbf{h}^{k-1}|\mathbf{h}^k)}{\partial \theta}. \quad (60)$$

An update of the top-level RBM is also performed in the wake phase, with $\mathbf{h}^{\ell-1}$ as observation for its visible vector. In the **sleep phase**, we generate a full observation $(\mathbf{h}^0, \mathbf{h}^1, \mathbf{h}^{\ell-1})$ from the model: we first sample $\mathbf{h}^{\ell-1}$ from the top-level RBM, and then sample each $\mathbf{h}^k$ according to $P(\mathbf{h}^k|\mathbf{h}^{k+1})$. This is then used as fully observed training data for the recognition conditionals $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$, by making a stochastic step in the direction of the following gradient:

$$\sum_{\mathbf{h}^0,\ldots,\mathbf{h}^{\ell-1}} P(\mathbf{h}^{\ell-1})P(\mathbf{h}^{\ell-2}|\mathbf{h}^{\ell-1}) \ldots P(\mathbf{h}^0|\mathbf{h}^1) \prod_{k=1}^{\ell-1} \frac{\partial \log Q(\mathbf{h}^k|\mathbf{h}^{k-1})}{\partial \theta}. \quad (61)$$

With respect to the log-likelihood gradient decomposition that we have been describing in this section, the approximations performed with the wake-sleep algorithm are thus the following: (a) approximate the gradient with respect to $KL(Q(h|x)||P(h|x))$ by the gradient with respect to $KL(P(h|x)||Q(h|x))$, and (b) approximate

$$\sum_h P(h|x) \log \frac{P(h|x)}{Q(h|x)}\frac{\partial \log P(h|x)}{\partial \theta} - \sum_h Q(h|x) \log \frac{P(h|x)}{Q(h|x)}\frac{\partial \log Q(h|x)}{\partial \theta} \approx 0 \quad (62)$$

which might be reasonable as long as $Q(h|x)$ is a good approximation of $P(h|x)$. Experiments suggest that the wake-sleep algorithm can be used (albeit slowly) to fine-tune a DBN and improve both the generative model and its ability to classify correctly (Hinton et al., 2006).

12 Global Optimization Strategies

Although deep architectures promise a more efficient representation of a distribution, and hence better generalization, they appear to come at the price of a more difficult optimization problem, as discussed earlier

in Section 5. Here, we draw connections between existing work and approaches that could help to deal this difficult optimization problem, based on the principle of **continuation methods** (Allgower & Georg, 1980). Although they provide no guarantee to obtain the global optimum, these methods have been particularly useful in computational chemistry to find approximate solutions of difficult optimization problems involving the configurations of molecules (Coleman & Wu, 1994; More & Wu, 1996; Wu, 1997). The basic idea is to first solve a smoothed version of the problem and gradually consider less smoothing, with the intuition that a smooth version of the problem reveals the global picture. On defines a single-parameter family of cost functions $C_\lambda(\theta)$ such that C_0 can be optimized 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 the dominant (if not global) minimum of C_1.

12.1 Greedy Layerwise Training of DBNs as a Continuation Method

The greedy layerwise training algorithm for DBNs described in Section 10 can be viewed as an approximate continuation method, as follows. First of all recall (Section 11.1) that an RBM (and in particular 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 layerwise procedure, we untie the parameters of the top-level RBM from the parameters of penultimate level. So one can view the layerwise procedure as follows. The model structure remains the same, an infinite chain of sigmoidal belief layers, but we change the constraint on the parameters at each step of the layerwise 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.

It would not be difficult to transform this layerwise approach into a continuation method by introducing a continuous parameter γ_k at each step d of adding a level to the DBN, such that when $\gamma_k = 0$ the parameters of the $(k+1)$-th level (and above) are still tied to those of the k-th, whereas when $\gamma_k = 1$, they are completely free of that constraint. But even in its current, discrete version, this analysis suggests an explanation for the good performance of the layerwise training approach in terms of reaching better optima, as evidenced in comparative experiments against the traditional optimization techniques in which all the levels are trained together (Bengio et al., 2007).

12.2 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* (Hastie, Rosset, Tibshirani, & Zhu, 2004) for an RBM, e.g., with ℓ_1 or ℓ_2 regularization of the parameters, the family of training criteria parametrized by $\lambda \in (0, 1]$:

$$C_\lambda(\theta) = -\sum_i \log P_\theta(x_i) - ||\theta||^2 \log \lambda. \tag{63}$$

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). Note that controlling the magnitude of the biases 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, stochastic gradient descent starting from small weights gradually allows the weights to increase in magnitude, 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 (Collobert & Bengio, 2004). There is no guarantee that the local minimum associated with each value of λ in eq. 63 is tracked by simply letting the weights follow the stochastic gradient path. It would not be difficult to slightly change the stochastic gradient algorithm to gradually increase 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.

12.3 Shaping: Training with a Curriculum

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 (Elman, 1993). 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** (Skinner, 1958; Peterson, 2004).

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 reweight the probability of sampling the examples from the distribution according to a given schedule, starting from the "easiest" examples and moving 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 $\hat{P}_t$). By making small changes in t correspond to smooth changes in the probability of sampling examples in the training distribution, we can construct a continuous path starting from an easy learning problem and ending in the desired training distribution.

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).

13 Other Comments

13.1 Deep + Distributed does not include only Neural

Although much of this paper has focused on deep neural net 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, as hinted at the end of Section 3.3.

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.

13.2 Why Sparse Representations and Not Dimensionality Reduction

We argue here that if one is going to have fixed-size representations (as in the brain), then sparse representations are more efficient to allow for varying number of bits per example. According to learning theory (Vapnik, 1995; Li & Vitanyi, 1997), 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 have different information content. This is why for example an image compression algorithm normally uses a different number of bits for different images (even if they all have the same dimensions).

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).

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 completely 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 Ranzato et al. (2008). This view actually explains how one could get good models even though the partition function is not explicitly maximized,

or only maximized approximately, as long as other constraints (such as sparsity) are used on the learned representation. Suppose that the representation learned by an autoassociator is sparse, then the autoassociator cannot reconstruct well every possible input pattern. To minimize the average reconstruction error on the training set, the autoassociator then has to find a representation which captures statistical regularities of the data distribution. First of all, Ranzato et al. (2008) 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 in eq. 13. Since the denominator (the partition function) is just a sum of the numerator over all possible input configurations, we would like to make reconstruction error high for most input configurations. 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 is high for most possible input patterns). One approach is to impose a sparsity penalty on the representation Ranzato et al. (2008), 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 RBM training, which only uses an *approximate* estimator of the gradient of the log of the partition function. If we add a penalty sparsity 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 configuations, and not only of the reconstructed neighbors of the input example that are obtained in the negative phase of Contrastive Divergence.

13.3 Other Reasons Why Unsupervised Learning is Crucial

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

- 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 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 disentangles them.

- Layer-wise unsupervised learning: this was argued in Section 5.3. 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 cases studied in Bengio et al. (2007).

- Less prone to overfitting: it has been argued (Hinton, 2006) that unsupervised learning is less prone to overfitting than supervised learning. The intuition is the following. When doing discriminant classification, one only needs to learn a function whose variations matter near the decision boundary.

A very small subset of the input variations might be relevant to uncover the proper classification. On the other hand, unsupervised learning tries to capture all the variations in the input. Therefore it requires a lot more capacity, or equivalently, is less prone to overfitting at equal capacity and equal number of training examples. Unsupervised learning can be used to initialize or regularize in the context of supervised learning systems.

- 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 local minima (those that do not correspond to good modeling of the input distribution).

14 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 achieve AI?
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, and if not how can that be fixed?
6. Is the presence of local minima an important issue in training RBMs?
7. Could we replace RBMs by algorithms that would be proficient at extracting good representations but involving an easier optimization problem, perhaps even a convex one?
8. Should the number of Gibbs steps in Contrastive Divergence be adjusted during training?
9. Besides reconstruction error, are there other more appropriate ways to monitor progress during training of RBMs? Equivalently, are there tractable approximations of the partition function in RBMs?
10. Could RBMs and autoassociators be improved by imposing some form of sparsity penalty on the representations they learn, and what would be good ways to do so?
11. Without increasing the number of hidden units, can the capacity of an RBM be increased using non-parametric forms of its energy function?
12. Is there a probabilistic interpretation to models learned in stacked autoassociators?
13. 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?
14. Can we obtain low variance and low bias estimators of the log-likelihood gradient in Deep Belief Networks, i.e., can we jointly train all the layers (with respect to the unsupervised objective)?
15. Can optimization strategies based on continuation methods deliver significantly improved training of Deep Belief Networks?
16. Aren't there other efficiently trainable deep architectures besides the Deep Belief Network model?

17. Is a curriculum needed to learn the kinds of high-level abstractions that humans take years or decades to learn?

18. 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?

19. Could we compute a tractable proxy for log-likelihood in Deep Belief Networks that could be used to monitor their performance during training, even in the unsupervised case?

20. 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)?

21. Although Deep Belief Networks are in principle well suited for the semi-supervised setting, how should their algorithms be adapted to this setting and how would they fare compared to existing semi-supervised algorithms?

22. When labeled examples are available, how should supervised and unsupervised criteria be combined to learn the model's representations of the input?

23. Can we find analogs of the computations necessary for Contrastive Divergence and Deep Belief Net learning in the brain?

24. Can decision tree ensembles be stacked to obtain and train a different type of deep architecture?

15 Conclusion

This paper 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 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 optimizing 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.
The paper focussed on a particular family of algorithms, the Deep Belief Networks, and their component elements, the Restricted Boltzmann Machine. 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 Autoassociators, based on a greedy, layerwise, 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.

Acknowledgements

The author is particularly grateful for the inspiration from and constructive discussions with Yann Le Cun, Geoffrey Hinton, Joseph Turian, Aaron Courville, Hugo Larochelle, Olivier Delalleau, Nicolas Le Roux, Jérôme Louradour, Pascal Lamblin, James Bergstra, and Dumitru Erhan. This research was performed thanks to funding from NSERC, MITACS, and the Canada Research Chairs.

References

Ackley, D., Hinton, G., & Sejnowski, T. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*.

Allgower, E. L., & Georg, K. (1980). *Numerical Continuation Methods. An Introduction*. No. 13 in Springer Series in Computational Mathematics. Springer-Verlag.

Andrieu, C., de Freitas, N., Doucet, A., & Jordan, M. (2003). An introduction to MCMC for machine learning. *Machine Learning*, *50*, 5–43.

Baxter, J. (1995). Learning internal representations. In *Proceedings of the Eighth International Conference on Computational Learning Theory*, pp. 311–320 Santa Cruz, California. ACM Press.

Baxter, J. (1997). A bayesian/information theoretic model of learning via multiple task sampling. *Machine Learning*, *28*, 7–40.

Belkin, M., & Niyogi, P. (2003). Using manifold structure for partially labeled classification. In Becker, S., Thrun, S., & Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15* Cambridge, MA. MIT Press.

Belkin, M., Matveeva, I., & Niyogi, P. (2004). Regularization and semi-supervised learning on large graphs. In Shawe-Taylor, J., & Singer, Y. (Eds.), *COLT'2004*. Springer.

Bell, A., & Sejnowski, T. (1995). An information maximisation approach to blind separation and blind deconvolution. *Neural Computation*, *7*(6), 1129–1159.

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. In Leen, T., Dietterich, T., & Tresp, V. (Eds.), *Advances in Neural Information Processing Systems 13*, pp. 933–938. MIT Press.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19*, pp. 153–160. MIT Press.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bengio, Y., & Delalleau, O. (2007). Justifying and generalizing contrastive divergence. Tech. rep. 1311, Dept. IRO, Université de Montréal.

Bengio, Y., Delalleau, O., & Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, pp. 107–114. MIT Press, Cambridge, MA.

Bengio, Y., Delalleau, O., Le Roux, N., Paiement, J.-F., Vincent, P., & Ouimet, M. (2004). Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, *16*(10), 2197–2219.

Bengio, Y., Delalleau, O., & Simard, C. (2007). Decision trees do not generalize to new variations. Tech. rep. 1304, Universite de Montreal, Dept. IRO.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, *3*, 1137–1155.

Bengio, Y., & Le Cun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., & Weston, J. (Eds.), *Large Scale Kernel Machines*. MIT Press.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152 Pittsburgh.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, *59*, 291–294.

Brand, M. (2003). Charting a manifold. In Becker, S., Thrun, S., & Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15*. MIT Press.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Carreira-Perpiñan, M., & Hinton, G. (2005). On contrastive divergence learning. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*.

Caruana, R. (1993). Multitask connectionist learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pp. 372–379.

Cohn, D., Ghahramani, Z., & Jordan, M. I. (1995). Active learning with statistical models. In Tesauro, G., Touretzky, D., & Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*. Cambridge MA: MIT Press.

Coleman, T. F., & Wu, Z. (1994). Parallel continuation-based global optimization for molecular conformation and protein folding. Tech. rep., Cornell University, Department of Computer Science.

Collobert, R., & Bengio, S. (2004). Links between perceptrons, mlps and svms. In *ICML '04: Twenty-first international conference on Machine learning* New York, NY, USA. ACM Press.

Cortes, C., Haffner, P., & Mohri, M. (2004). Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, *5*, 1035–1062.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Cristianini, N., Shawe-Taylor, J., Elisseeff, A., & Kandola, J. (2002). On kernel-target alignment. *Advances in Neural Information Processing Systems*, *14*, 367–373.

Cucker, F., & Grigoriev, D. (1999). Complexity lower bounds for approximation algebraic computation trees. *Journal of Complexity*, *15*(4), 499–512.

Dayan, P., Hinton, G., Neal, R., & Zemel, R. (1995). The Helmholtz machine. *Neural Computation*, *7*, 889–904.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407.

Delalleau, O., Bengio, Y., & Le Roux, N. (2005). Efficient non-parametric function induction in semi-supervised learning. In Cowell, R., & Ghahramani, Z. (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pp. 96–103. Society for Artificial Intelligence and Statistics.

Doi, E., Balcan, D. C., & Lewicki, M. S. (2006). A theoretical analysis of robust coding over noisy overcomplete channels. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, pp. 307–314. MIT Press, Cambridge, MA.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small.. *Cognition*, *48*, 781–799.

Freund, Y., & Haussler, D. (1994). Unsupervised learning of distributions on binary vectors using two layer networks. Tech. rep. UCSC-CRL-94-25, University of California, Santa Cruz.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pp. 148–156 USA. ACM.

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

Gärtner, T. (2003). A survey of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, *5*(1).

Geman, S., & Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*.

Hastad, J. (1986). 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.

Hastad, J., & Goldmann, M. (1991). On the power of small-depth threshold circuits. *Computational Complexity*, *1*, 113–129.

Hastie, T., Rosset, S., Tibshirani, R., & Zhu, J. (2004). The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, *5*, 1391–1415.

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

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA.

Hinton, G. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1–12 Amherst 1986. Lawrence Erlbaum, Hillsdale.

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*, 1771–1800.

Hinton, G. (2006). To recognize shapes, first learn to generate images. Tech. rep. UTML TR 2006-003, University of Toronto.

Hinton, G., Dayan, P., Frey, B., & Neal, R. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, *268*, 1558–1161.

Hinton, G., Sejnowski, T., & Ackley, D. (1984). Boltzmann machines: Constraint satisfaction networks that learn. Tech. rep. TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science.

Hinton, G., Welling, M., Teh, Y., & Osindero, S. (2001). A new view of ica. In *Proceedings of ICA-2001* San Diego, CA.

Hinton, G. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)*, Vol. 1, pp. 1–6.

Hinton, G. E., & Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*, 504–507.

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length, and Helmholtz free energy. *Advances in Neural Information Processing Systems*, *6*, 3–10.

Ho, T. K. (1995). Random decision forest. In *3rd International Conference on Document Analysis and Recognition*, pp. 278–282 Montreal, Canada.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.. See www7.informatik.tu-muenchen.de/˜hochreit.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*, 417–441, 498–520.

Hubel, D., & Wiesel, T. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology (London)*, *160*, 106–154.

Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin.

Intrator, N., & Edelman, S. (1996). How to make a low-dimensional representation suitable for diverse tasks. *Connection Science, Special issue on Transfer in Neural Networks*, *8*, 205–224.

Jaakkola, T., & Haussler, D. (1998). Exploiting generative models in discriminative classifiers..

Japkowicz, N., Hanson, S. J., & Gluck, M. A. (2000). Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, *12*(3), 531–545.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, *1*(1), 1–7.

Lanckriet, G., Cristianini, N., Bartlett, P., El Gahoui, L., & Jordan, M. (2002). Learning the kernel matrix with semi-definite programming. In *ICML'2002*.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Twenty-fourth International Conference on Machine Learning (ICML'2007)*.

Le Roux, N., & Bengio, Y. (2008). Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, *to appear*.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

LeCun, Y., Bottou, L., Orr, G., & Müller, K.-R. (1998a). Efficient backprop. In Orr, G., & Müller, K.-R. (Eds.), *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998b). Gradient based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M.-A., & Huang, F.-J. (2006). A tutorial on energy-based learning. In Bakir, G., Hofman, T., Scholkopf, B., Smola, A., & Taskar, B. (Eds.), *Predicting Structured Data*. MIT Press.

LeCun, Y., & Huang, F. (2005). Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05)*.

Lewicki, M., & Sejnowski, T. (1998). Learning nonlinear overcomplete representations for efficient coding. In Jordan, M., Kearns, M., & Solla, S. (Eds.), *Advances in Neural Information Processing Systems 10*. MIT Press.

Li, M., & Vitanyi, P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications*. Second edition, Springer, New York, NY.

Lin, T., Horne, B., Tino, P., & Giles, C. (1995). Learning long-term dependencies is not as difficult with NARX recurrent neural networks. Tech. rep. UMICAS-TR-95-78, Institute for Advanced Computer Studies, University of Mariland.

McCulloch, W., & Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*.

Memisevic, R., & Hinton, G. (2007). Unsupervised learning of image transformations. In *CVPR'07: Proceedings of the 2007 Conference on Computer Vision and Pattern Recognition*.

Mendelson, E. (1997). *Introduction to Mathematical Logic, 4th ed.* Chapman & Hall.

Miikkulainen, R., & Dyer, M. (1991). Natural language processing with modular pdp networks and distributed lexicon. *Cognitive Science, 15*, 343–399.

Mnih, A., & Hinton, G. E. (2007). Three new graphical models for statistical language modelling. In Ghahramani, Z. (Ed.), *Twenty-fourth International Conference on Machine Learning (ICML'2007)*, pp. 641–648. Omnipress.

More, J., & Wu, Z. (1996). Smoothing techniques for macromolecular global optimization. In Pillo, G. D., & Giannessi, F. (Eds.), *Nonlinear optimization and applications*. Plenum Press.

Olshausen, B., & Field, D. (1997). Sparse coding with an overcomplete basis set: a strategy employed by V1?. *Journal Research, 37*, 3311–3325.

Orponen, P. (1994). Computational complexity of neural networks: a survey. *Nordic Journal of Computing, 1*(1), 94–110.

Osindero, S., & Hinton, G. E. (2008). Modeling image patches with a directed hierarchy of markov random field. In *Neural Information Processing Systems Conference (NIPS) 20*.

Pearlmutter, B., & Parra, L. (1996). A context-sensitive generalization of ICA. In Xu, L. (Ed.), *International Conference On Neural Information Processing* Hong-Kong.

Pérez, E., & Rendell, L. A. (1996). Learning despite concept variation by finding structure in attribute-based data. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 391–399.

Peterson, G. B. (2004). A day of great illumination: B.F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior, 82*(3), 317–328.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence, 46*(1), 77–105.

Rabiner, L., & Juang, B. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, 257–285.

Ranzato, M., Boureau, Y., Chopra, S., & LeCun, Y. (2007). A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)*.

Ranzato, M., Boureau, Y.-L., & LeCun, Y. (2008). Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*. MIT Press.

Ranzato, M., Huang, F., Boureau, Y., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press.

Ranzato, M., & LeCun, Y. (2007). A sparse and locally shift invariant feature extractor applied to document images. In *International Conference on Document Analysis and Recognition (ICDAR)*.

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In et al., J. P. (Ed.), *Advances in Neural Information Processing Systems (NIPS 2006)*. MIT Press.

Rissanen, J. (1990). *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

Rumelhart, D., Hinton, G., & Williams, R. (1986a). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Rumelhart, D., McClelland, J., & the PDP Research Group (1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1. MIT Press, Cambridge.

Salakhutdinov, R., & Hinton, G. (2007). Semantic hashing. In *Proceedings of the 2007 Workshop on Information Retrieval and applications of Graphical Models (SIGIR 2007)*.

Salakhutdinov, R., & Hinton, G. (2008). Using deep belief nets to learn covariance kernels for gaussian processes. In Platt, J., Koller, D., Singer, Y., & Roweis, S. (Eds.), *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA.

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

Schölkopf, B., Burges, C. J. C., & Smola, A. J. (1999a). *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., & Smola, A. (1999b). Input space versus feature space. *IEEE Trans. Neural Networks*, *100*(5), 1000–1017.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*, 1299–1319.

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

Schwenk, H., & Milgram, M. (1995). Transformation invariant autoassociation with application to handwritten character recognition. In Tesauro, G., Touretzky, D., & Leen, T. (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 991–998. MIT Press.

Schwenk, H. (2004). Efficient training of large neural networks for language modeling. In *International Joint Conference on Neural Networks*.

Schwenk, H., & Gauvain, J.-L. (2005). Building continuous space language models for transcribing european languages. In *Interspeech*, pp. 737–740.

Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., & Poggio, T. (2007). A quantitative theory of immediate visual recognition. *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, *165*, 33–56.

Simard, P.Y. Steinkraus, D., & Platt, J. (2003). Best practices for convolutional neural networks. In *Proc. of ICDAR*.

Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, *13*, 94–99.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D., & McClelland, J. (Eds.), *Parallel Distributed Processing*, Vol. 1, chap. 6, pp. 194–281. MIT Press, Cambridge.

Solomonoff, R. J. (1964). A formal theory of inductive inference. *Information and Control, 7*, 1–22, 224–254.

Sutskever, I., & Hinton, G. (2007). Learning multilevel distributed representations for high-dimensional sequences. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, March 21-24, 2007, Porto-Rico*.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Taylor, G., Hinton, G., & Roweis, S. (2006). Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems 20*. MIT Press.

Teh, Y.-W., Welling, M., Osindero, S., & Hinton, G. E. (2003). Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, *4*, 1235–1260.

Tenenbaum, J., de Silva, V., & Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323.

Titov, I., & Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Proc. 45th Meeting of Association for Computational Linguistics (ACL 2007)* Prague, Czech Republic.

Utgoff, P., & Stracuzzi, D. (2002). Many-layered learning. *Neural Computation*, *14*, 2497–2539.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

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

Wallace, C., & Boulton, D. (1968). An information measure for classification. *Computer Journal*, *11*(2), 185–194.

Wang, L., & Luk Chan, K. (2002). Learning kernel parameters by using class separability measure. In *proc. NIPS*.

Wegener, I. (1987). *The Complexity of Boolean Functions*. John Wiley & Sons.

Weiss, Y. (1999). Segmentation using eigenvectors: a unifying view. In *Proceedings IEEE International Conference on Computer Vision*, pp. 975–982.

Welling, M., Rosen-Zvi, M., & Hinton, G. (2005). Exponential family harmoniums with an application to information retrieval. In Saul, L., Weiss, Y., & Bottou, L. (Eds.), *Advances in Neural Information Processing Systems 17*. MIT Press.

Welling, M., Zemel, R., & Hinton, G. E. (2003). Self-supervised boosting. In Becker, S., Thrun, S., & Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15*. MIT Press.

Williams, C., & Rasmussen, C. (1996). Gaussian processes for regression. In Touretzky, D., Mozer, M., & Hasselmo, M. (Eds.), *Advances in Neural Information Processing Systems 8*, pp. 514–520. MIT Press, Cambridge, MA.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*, 241–249.

Wu, Z. (1997). Global continuation for distance geometry problems. *SIAM Journal of Optimization*, *7*, 814–836.

Xu, P., Emami, A., & Jelinek, F. (2003). Training connectionist models for the structured language model. In *Empirical Methods in Natural Language Processing, EMNLP'2003*.

Yao, A. (1985). Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pp. 1–10.

Zhou, D., Bousquet, O., Navin Lal, T., Weston, J., & Schölkopf, B. (2004). Learning with local and global consistency. In Thrun, S., Saul, L., & Schölkopf, B. (Eds.), *Advances in Neural Information Processing Systems 16* Cambridge, MA. MIT Press.

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML'2003*.