On the Expressive Power of Deep Architectures

Yoshua Bengio and Olivier Delalleau

Dept. IRO, Université de Montréal. Montréal (QC), H3C 3J7, Canada

Abstract. Deep architectures are families of functions corresponding to deep circuits. Deep Learning algorithms are based on parametrizing such circuits and tuning their parameters so as to approximately optimize some training objective. Whereas it was thought too difficult to train deep architectures, several successful algorithms have been proposed in recent years. We review some of the theoretical motivations for deep architectures, as well as some of their practical successes, and propose directions of investigations to address some of the remaining challenges.

1 Learning Artificial Intelligence

An intelligent agent takes good decisions. In order to do so it needs some form of knowledge. Knowledge can be embodied into a function that maps inputs and states to states and actions. If we saw an agent that always took what one would consider as the good decisions, we would qualify the agent as intelligent. Knowledge can be explicit, as in the form of symbolically expressed rules and facts of expert systems, or in the form of linguistic statements in an encyclopedia. However, knowledge can also be implicit, as in the complicated wiring and synaptic strengths of animal brains, or even in the mechanical properties of an animal's body. Whereas Artificial Intelligence (AI) research initially focused on providing computers with knowledge in explicit form, it turned out that much of our knowledge was not easy to express formally. What is a *chair*? We might write a definition that can help another human understand the concept (if he did not know about it), but it is difficult to make it sufficiently complete for a computer to translate into the same level of competence (e.g. in recognizing chairs in images). Much so-called *common-sense knowledge* has this property.

If we cannot endowe computers with all the required knowledge, an alternative is to let them learn it from examples. Machine learning algorithms aim to extract knowledge from examples (i.e., data), so as to be able to properly generalize to new examples. Our own implicit knowledge arises either out of our life experiences (lifetime learning) or from the longer scale form of learning that evolution really represents, where the result of adaptation is encoded in the genes. Science itself is a process of learning from observations and experiments in order to produce actionable knowledge. Understanding the principles by which agents can capture knowledge through examples, i.e., learn, is therefore a central scientific question with implications not only for AI and technology, but also to understand brains and evolution.

Formally, a learning algorithm can be seen as a functional that maps a dataset (a set of examples) to a function (typically, a decision function). Since the dataset is itself a random variable, the learning process involves the application of a procedure to a target distribution from which the examples are drawn and for which one would like to infer a good decision function. Many modern learning algorithms are expressed as an optimization problem, in which one tries to find a compromise between minimizing empirical error on training examples and minimizing a proxy for the richness of the family of functions that contains the solution. A particular challenge of learning algorithms for AI tasks (such as understanding images, video, natural language text, or speech) is that such tasks involve a large number of variables with complex dependencies, and that the amount of knowledge required to master these tasks is very large. Statistical learning theory teaches us that in order to represent a large body of knowledge, one requires a correspondingly large number of degrees of freedom (or richness of a class of functions) and a correspondingly large number of training examples. In addition to the statistical challenge, machine learning often involves a computational challenge due to the difficulty of optimizing the training criterion. Indeed, in many cases, that training criterion is not convex, and in some cases it is not even directly measurable in a deterministic way and its gradient is estimated by stochastic (sampling-based) methods, and from only a few examples at a time (online learning).

One of the characteristics that has spurred much interest and research in recent years is **depth of the architecture**. In the case of a multi-layer neural network, depth corresponds to the number of (hidden and output) layers. A fixed-kernel Support Vector Machine is considered to have depth 2 (Bengio and LeCun, 2007a) and boosted decision trees to have depth 3 (Bengio *et al.*, 2010). Here we use the word *circuit* or *network* to talk about a directed acyclic graph, where each node is associated with some output value which can be computed based on the values associated with its predecessor nodes. The arguments of the learned function are set at the input nodes of the circuit (which have no predecessor) and the outputs of the function are read off the output nodes of the circuit. Different families of functions correspond to different circuits and allowed choices of computations in each node. Learning can be performed by changing the computation associated with a node, or rewiring the circuit (possibly changing the number of nodes). The depth of the circuit is the length of the longest path in the graph from an input node to an output node.

This paper also focuses on **Deep Learning**, i.e., learning *multiple levels of representation*. The intent is to discover more *abstract* features in the higher levels of the representation, which hopefully make it easier to *separate from each other the various explanatory factors extent in the data*. Theoretical results (Yao, 1985; Håstad, 1986; Håstad and Goldmann, 1991; Bengio *et al.*, 2006; Bengio and Delalleau, 2011; Braverman, 2011), reviewed briefly here (see also a previous discussion by Bengio and LeCun, 2007b) 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) associated with functions with many variations but an underlying simpler structure, one may need *deep architectures*. The recent surge in experimental work in the field seems to support this notion, accumulating evidence that in challenging AI-related tasks – such as computer vision (Bengio *et al.*, 2007; Ranzato *et al.*, 2007; Larochelle *et al.*, 2007; Ranzato *et al.*, 2008; Lee *et al.*, 2009; Mobahi *et al.*, 2009; Osindero and Hinton, 2008), natural language processing (NLP) (Collobert and Weston, 2008a; Weston *et al.*, 2008), robotics (Hadsell *et al.*, 2008), or information retrieval (Salakhutdinov and Hinton, 2007; Salakhutdinov *et al.*, 2007) – deep learning methods significantly out-perform comparable but shallow competitors (e.g. winning the Unsupervised and Transfer Learning Challenge; Mesnil *et al.*, 2011), and often match or beat the state-of-the-art.

In this paper we discuss some of the theoretical motivations for deep architectures, and quickly review some of the current layer-wise unsupervised feature-learning algorithms used to train them. We conclude with a discussion of principles involved, challenges ahead, and ideas to face them.

2 Local and Non-Local Generalization: The Challenge and Curse of Many Factors of Variation

How can learning algorithms generalize from training examples to new cases? It can be shown that there are no completely universal learning procedures, in the sense that for any learning procedure, there is a target distribution on which it does poorly (Wolpert, 1996). Hence, all generalization principles exploit some property of the target distribution, i.e., some kind of prior. The most exploited generalization principle is that of *local generalization*. It relies on a *smoothness assumption*, i.e., that the target function (the function to be learned) is smooth (according to some measure of smoothness), i.e., changes slowly and rarely (Barron, 1993). Contrary to what has often been said, what mainly hurts many algorithms relying only on this assumption (pretty much all of the non-parametric statistical learning algorithms) is not the dimensionality of the input but instead the insufficient smoothness of the target function[1].

To make a simple picture, imagine the supervised learning framework and a target function that is locally smooth but has many ups and downs in the domain of interest. We showed that if one considers a straight line in the input domain, and counts the number of ups and downs along that line, then a learner based purely on local generalization (such as a Gaussian kernel machine) requires at least as many examples as there are ups and downs (Bengio *et al.*, 2006).

Manifold learning algorithms are unsupervised learning procedures aiming to characterize a low-dimensional manifold near which the target distribution concentrates. Bengio and Monperrus (2005) argued that many real-world manifolds (such as the one generated by translations or rotations of images, when the image is represented by its pixel intensities) are highly curved (translating by 1

[1] but of course additional noisy dimensions, although they do not change smoothness of the target function, require more examples to cancel the noise.

pixel can change the tangent plane of the manifold by about 90 degrees). The manifold learning algorithms of the day, based implicitly or explicitly on non-parametric estimation of the local tangent planes to the manifold, are relying on purely local generalization. Hence they would require a number of examples that grows linearly with the dimension d of the manifold and the number of patches $O\left(\frac{D}{r}\right)^d$ needed to cover its nooks and crannies, i.e., in $O\left(d\left(\frac{D}{r}\right)^d\right)$ examples, where D is a diameter of the domain of interest and r a radius of curvature.

3 Expressive Power of Deep Architectures

To fight an exponential, it seems reasonable to arm oneself with other exponentials. We discuss two strategies that can bring a potentially exponential statistical gain thanks to a combinatorial effect: distributed (possibly sparse) representations and depth of architecture. We also present an example of the latter in more details in the specific case of so-called sum-product networks.

3.1 Distributed and Sparse Representations

Learning algorithms based on *local generalization* can generally be interpreted as creating a number of *local regions* (possibly overlapping, possibly with soft rather than hard boundaries), such that each region is associated with its own degrees of freedom (parameters, or examples such as prototypes). Such learning algorithms can then learn to discriminate between these regions, i.e., provide a different response in each region (and possibly doing some form of smooth interpolation when the regions overlap or have soft boundaries). Examples of such algorithms include the mixture of Gaussians (for density estimation), Gaussian kernel machines (for all kinds of tasks), ordinary clustering (such as k-means, agglomerative clustering or affinity propagation), decision trees, nearest-neighbor and Parzen windows estimators, etc... As discussed in previous work (Bengio *et al.*, 2010), all of these algorithms will generalize well only to the extent that there are enough examples to cover all the regions that need to be distinguished from each other.

As an example of such algorithms, the way a clustering algorithm or a nearest-neighbor algorithm could partition the input space is shown on the left side of Fig. 1. Instead, the right side of the figure shows how an algorithm based on distributed representations (such as a Restricted Boltzmann Machine; Hinton *et al.*, 2006) could partition the input space. Each binary hidden variable identifies on which side of a hyper-plane the current input lies, thus breaking out input space in a number of regions that could be exponential in the number of hidden units (because one only needs a few examples to learn where to put each hyper-plane), i.e., in the number of parameters. If one assigns a binary code to each region, this is also a form of clustering, which has been called *multi-clustering* (Bengio, 2009).

Distributed representations were put forward in the early days of connectionism and artificial neural networks (Hinton, 1986, 1989). More recently, a

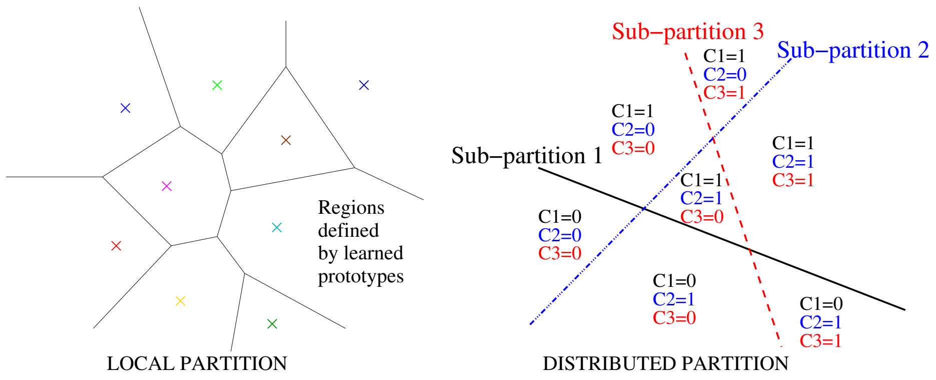

Fig. 1. Contrast beween learning algorithms such as clustering (left side), based on local generalization, with examples required in each input region distinguished by the learner, and algorithms based on distributed representations, such as a Restricted Boltzmann Machine (right side). The latter also splits up the input space in regions but where the number of parameters or examples required can be much smaller (potentially exponentially smaller) than the number of regions one can distinguish. This is what grants the possibility of generalizing to regions where no data have been observed.

variation on distributed representations has been explored by many researchers, which is somehow in between purely local representations and the traditional dense distributed representations: sparse representations. The idea is that only a few dimensions of the representations are "active", with the inactive dimensions basically set to 0 or close to 0. Neurons in the cortex are believed to have a distributed and sparse representation (Olshausen and Field, 1997), with around 1-4% of the neurons active at any one time (Attwell and Laughlin, 2001; Lennie, 2003). With k out of d active dimensions in a representation, one still gets (potentially) exponentially more representational power than a local representation, with the number of regions that can be distinguished now being in the order of n choose k. See Bengio (2009) for a brief overview of the literature on sparse representations.

3.2 Depth

Depth is a notion borrowed from complexity theory, and that is defined for *circuits*. A circuit is a directed acyclic graph where each node is associated with a computation, and whose output results are used by the successors of that node. Input nodes have no predecessor and output nodes have no successor. The depth of a circuit is the longest path from an input to an output node. A long-standing question in complexity theory is the extent to which depth-limited circuits can represent functions as efficiently as deeper circuits. A depth-2 circuit (with appropriate choice of computational elements, e.g. logic gates or formal neurons) can compute or approximate any function, but it may require an exponentially large number of nodes. This is a relevant question for machine learning, because many learning algorithms learn "shallow architectures" (Bengio and LeCun, 2007b), typically of depth 1 (linear predictors) or 2 (most non-parametric predictors). If AI tasks require deeper circuits (and human brains certainly appear deep), then we should find ways to incorporate depth into our learning algorithms. The consequences of using a too shallow predictor would be that it may not generalize well, unless given huge numbers of examples and capacity (i.e., computational resources and statistical resources).

The early results on the limitations of shallow circuits regard functions such as the parity function (Yao, 1985), showing that logic gates circuits of depth-2 require exponential size to implement d-bit parity where a deep circuit of depth $O(\log(d))$ could implement it with $O(d)$ nodes. Håstad (1986) then showed that there are functions computable with a polynomial-size logic gate circuit of depth k that require exponential size when restricted to depth $k-1$ (Håstad, 1986). Interestingly, a similar result was proven for the case of circuits made of linear threshold units (formal neurons; Håstad and Goldmann, 1991), when trying to represent a particular family of functions. A more recent result brings an example of a very large class of functions that cannot be efficiently represented with a small-depth circuit (Braverman, 2011). It is particularly striking that the main theorem regards the representation of functions that capture dependencies in joint distributions. Basically, dependencies that involve more than r variables are difficult to capture by shallow circuits. An r-independent distribution is one that cannot be distinguished from the uniform distribution when looking only at r variables at a time. The proof of the main theorem (which concerns distribution over bit vectors) relies on the fact that order-r polynomials over the reals cannot capture r-independent distributions. The main result is that bounded-depth circuits cannot distinguish data generated by r-independent distributions from independent noisy bits. We have also recently shown (Bengio and Delalleau, 2011) results for *sum-product networks* (where nodes either compute sums or products, over the reals). We present these results in more details below as an example of the advantage brought by depth in terms of the efficiency of the representation: we found two families of polynomials that can be efficiently represented with depth-d circuits, but require exponential size with depth-2 circuits. Interestingly, sum-product networks were recently proposed to efficiently represent high-dimensional joint distributions (Poon and Domingos, 2011).

Besides the complexity-theory hints at their representational advantages, there are other motivations for studying learning algorithms which build a deep architecture. The earliest one is simply inspiration from brains. By putting together anatomical knowledge and measures of the time taken for signals to travel from the retina to the frontal cortex and then to motor neurons (about 100 to 200ms), one can gather that at least 5 to 10 feedforward levels are involved for some of the simplest visual object recognition tasks. Slightly more complex vision tasks require iteration and feedback top-down signals, multiplying the overall depth by an extra factor of 2 to 4 (to about half a second).

Another motivation derives from what we know of cognition and abstractions: as argued by Bengio (2009), it is natural for humans to represent concepts at one level of abstraction as the *composition* of concepts at lower levels. Engineers often craft representations at multiple levels, with higher levels obtained by transformation of lower levels. Instead of a flat `main` program, software engineers structure their code to obtain plenty of *re-use*, with functions and modules re-using other functions and modules. This inspiration is directly linked to machine learning: deep architectures appear well suited to *represent higher-level abstractions* because they lend themselves to *re-use*. For example, some of the features that are useful for one task may be useful for another, making Deep Learning particularly well suited for *transfer learning* and *multi-task learning* (Caruana, 1995; Collobert and Weston, 2008b; Bengio *et al.*, 2011; Bengio, 2011). Here one is exploiting the existence of underlying common explanatory factors that are useful for multiple tasks. This is also true of *semi-supervised learning*, which exploits connections between the input distribution $P(X)$ and a target conditional distribution $P(Y|X)$ (see Weston *et al.* (2008) for a first application of Deep Learning to semi-supervised learning). In general these two distributions, seen as functions of x, may be unrelated to each other. But in the world around us, it is often the case that some of the factors that shape the input variables X are predictive of the output variables Y. Deep Learning relies heavily on unsupervised or semi-supervised learning, and assumes that *representations of* X *that are useful to capture* $P(X)$ *are also in part useful to capture* $P(Y|X)$. An extensive study by Erhan *et al.* (2010) has explored the question of whether and how this prior may explain the success of the *greedy layer-wise unsupervised pre-training* recipe followed in many Deep Learning algorithms, and explained in Sect. 4.

3.3 A Deep Sum-Product Networks Case Study

Poon and Domingos (2011) introduced deep **sum-product networks** as a method to compute partition functions of tractable graphical models. These networks are analogous to traditional artificial neural networks but with nodes that compute either products or weighted sums of their inputs. In this setting the advantage brought by depth may not be obvious: after all, the output value can always be written as a sum of products of input variables (possibly raised to some power), and consequently it is easily rewritten as a shallow network with a sum output unit and product hidden units.

The argument supported by our theoretical analysis (Bengio and Delalleau, 2011) is that a deep architecture is able to compute some functions much more efficiently than a shallow one. Here we measure “efficiency” in terms of the number of computational units in the network. Bengio (2009) suggested that some polynomials could be represented more efficiently by deep sum-product networks, but without providing any formal statement or proofs. We partly addressed this void by demonstrating families of circuits for which a deep architecture can be exponentially more efficient than a shallow one in the context of real-valued polynomials[2].

In the following we briefly review our main results, in which we consider two families of functions represented by deep sum-product networks (denoted by $\mathcal{F}$ and $\mathcal{G}$). For each family, we establish a lower bound on the minimal number of hidden units a shallow (depth-2) sum-product network would require to represent a function of this family, showing it is much less efficient than the deep representation.

The first family of functions we study is $\mathcal{F} = \cup_{n \geq 4} \mathcal{F}_n$, where $\mathcal{F}_n$ is made of functions built from deep sum-product networks with $n = 2^k$ inputs and (even) depth k that alternate binary product and sum layers (Fig. 2 for the simplest case, $\mathcal{F}_4$).

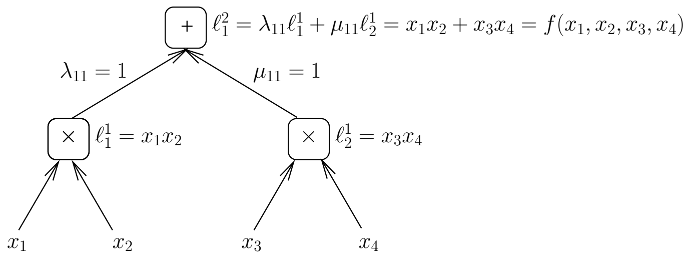

Fig. 2. Sum-product network computing some $f \in \mathcal{F}_4$, i.e., with n=4 inputs and depth $k = \log_2 n = 2$.

The second family of functions is $\mathcal{G} = \cup_{n \geq 2, i \geq 0} \mathcal{G}_{in}$ such that the sub-family $\mathcal{G}_{in}$ is made of sum-product networks with n input variables and depth $2i + 1$, that alternate sum and product layers. Each sum or product unit takes $n - 1$ units from the previous layer as inputs. An example of a network belonging to $\mathcal{G}_{1,3}$ is shown in Fig. 3 (it has unit summation weights to keep the figure easy to read). Note that contrary to family $\mathcal{F}$, depth and input size can be varied independently for networks in $\mathcal{G}$.

The main result for family $\mathcal{F}$ is that any shallow sum-product network computing a function in $\mathcal{F}_n$ must have at least $2^{\sqrt{n}-1}$ hidden units. The high-level proof sketch consists in the following steps (Bengio and Delalleau, 2011):

1. Show that the number of unique products found in the expanded polynomial representation of $f \in \mathcal{F}_n$ is $2^{\sqrt{n}-1}$.
2. Prove that the only possible architecture for a shallow sum-product network to compute f is to have a hidden layer made of product units, with a sum unit as output.

[2] Here we restrict our definition of "sum-product networks" to those networks whose summation units have positive incoming weights, even though some of our results still hold for networks with non-positive weights (Bengio and Delalleau, 2011).

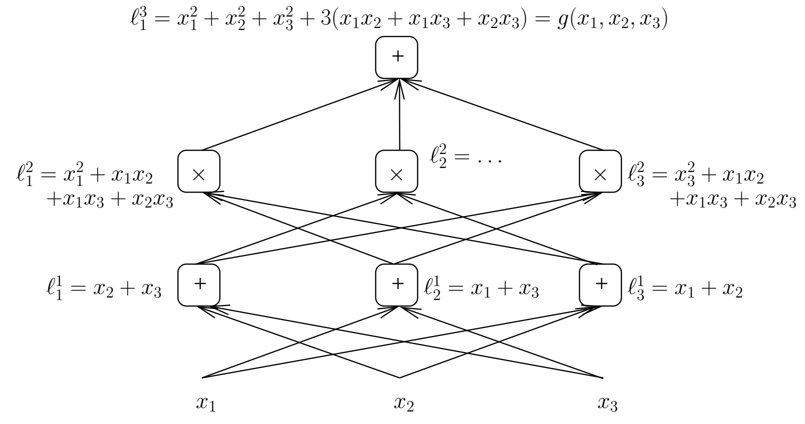
$\ell_1^3 = x_1^2 + x_2^2 + x_3^2 + 3(x_1x_2 + x_1x_3 + x_2x_3) = g(x_1, x_2, x_3)$
$+$
$\ell_1^2 = x_1^2 + x_1x_2 + x_1x_3 + x_2x_3$
$\times$
$\ell_2^2 = \dots$
$\times$
$\times$
$\ell_3^2 = x_3^2 + x_1x_2 + x_1x_3 + x_2x_3$
$\ell_1^1 = x_2 + x_3$
$+$
$+$
$\ell_2^1 = x_1 + x_3$
$+$
$\ell_3^1 = x_1 + x_2$
x_1
x_2
x_3

Fig. 3. Sum-product network computing some $g \in \mathcal{G}_{1,3}$.

3. Conclude that the number of hidden units in step 2 must be at least the number of unique products computed in step 1.

For family $\mathcal{G}$, we obtain that a shallow sum-product network computing $g_{in} \in \mathcal{G}_{in}$ must have at least $(n-1)^i$ hidden units. The proof relies on a similar idea, i.e. we use a lower bound on the number of products found in the expanded polynomial expansion of g to bound the number of hidden units in a shallow sum-product network with summation output. In addition, the final result uses the degree of the output polynomial, which is $(n-1)^i$, to bound the number of hidden units in a shallow sum-product network with product output (also proving there can be no product units in the hidden layer).

In summary, we obtain that **functions in families $\mathcal{F}$ and $\mathcal{G}$ can be computed by a deep sum-product network with exponentially less units than when computed by a shallow sum-product network**. This motivates using deep sum-product networks to obtain more efficient representations.

4 A Zoo of Learning Algorithms

Greedy Layer-Wise Unsupervised Feature Learning.
Whereas early efforts at training deep architectures were unsuccessful (Bengio and LeCun, 2007a), a major breakthrough in Deep Learning methods came about with the use of *layer-wise unsupervised learning* (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007), as a way to initialize a deep supervised neural network.

Deep Learning usually occurs in two phases: first, unsupervised, layer-wise training, and second, supervised training of a classifier that exploits what has been done in the first phase. In the unsupervised phase, each layer is added and trained greedily, i.e., keeping the earlier layers fixed and ignoring the future interactions with additional layers. Each layer uses the representation learned by the previous layer as input that it tries to model and transform to a new and better representation. Many unsupervised learning algorithms are being explored for the first phase, including various methods to train Restricted Boltzmann Machines (RBMs) (Freund and Haussler, 1994; Hinton *et al.*, 2006; Tieleman, 2008; Salakhutdinov and Hinton, 2009; Desjardins *et al.*, 2010) or Deep Boltzmann Machines (Salakhutdinov and Hinton, 2010; Lee *et al.*, 2009), different flavours of auto-encoders (Bengio *et al.*, 2007; Vincent *et al.*, 2008; Larochelle *et al.*, 2009), and other sparse encoder-decoder systems (Ranzato *et al.*, 2007; Kavukcuoglu *et al.*, 2009).

The objective stated in the Deep Learning literature is to discover powerful representation-learning algorithms, mostly thanks to unsupervised learning procedures. Ideally, such representations should somehow capture the salient factors of variation that explain the data, and this can be tested by attempting to use these learned representations to predict some of these factors, e.g., in a classification problem.

Boltzmann Machines. The first unsupervised learning algorithm (Hinton and Salakhutdinov, 2006; Hinton *et al.*, 2006) that has been proposed for training each layer of a deep architecture is based on a Restricted Boltzmann Machine (Smolensky, 1986), which is an undirected graphical model that is a particular form of Boltzmann Machine (Hinton *et al.*, 1984). A Boltzmann Machine is an undirected graphical model for observed variable x based on latent variable h is specified by an *energy function* $\mathbb{E}(x, h)$:

$$P(x, h) = \frac{e^{-\mathbb{E}(x,h)}}{Z}$$

where Z is a normalization constant called the partition function. A Boltzmann machine is one where $\mathbb{E}(x, h)$ is a second-order polynomial in (x, h), e.g.,

$$\mathbb{E}(x, h) = h'Wx + h'Uh + x'Vx + b'h + c'x$$

and in general both x and h are considered to be binary vectors, which makes Z intractable except when both x and h have very few components. The coefficients $\theta = (W, U, V, b, c)$ of that second-order polynomial are the parameters of the model. Given an observed x, the inference $P(h|x)$ is generally intractable but can be estimated by sampling from a Monte-Carlo Markov Chain (MCMC), e.g. by Gibbs sampling, or using loopy belief, variational or mean-field approximations. Even though computing the energy is easy, marginalizing over h in order to compute the likelihood $P(x)$ is generally intractable, so that the exact log-likelihood gradient is also intractable. However, several algorithms have been proposed in recent years to estimate the gradient, most of them based on the following decomposition into the so-called "positive phase part" (x is fixed to the observed value, the gradient term tends to decrease the associated energy) and "negative phase part" (both x and h are sampled according to P, and the gradient term tends to increase their energy):

$$\frac{\partial}{\partial \theta}(-\log P(x)) = E_h \left[\frac{\partial \mathbb{E}(x, h)}{\partial \theta} | x \right] - E_{x,h} \left[\frac{\partial \mathbb{E}(x, h)}{\partial \theta} \right].$$

Even though a Boltzmann Machine is a parametric model when we consider the dimensionality n_h of h to be fixed, in practice one allows n_h to vary, making it a non-parametric model. With n_h large enough, one can model any discrete distribution: Le Roux and Bengio (2008) showed that Restricted Boltzmann Machines (described below) are universal approximators, and since they are special cases of Boltzmann Machines, Boltzmann Machines also are universal approximators. On the other hand with $n_h > 0$ the log-likelihood is not anymore convex in the parameters, and training can potentially get stuck in one of many local minima.

The Restricted Boltzmann Machine (RBM) is a Boltzmann machine without lateral interactions, i.e., $U = 0$ and $V = 0$. It turns out that the positive phase part of the gradient can be computed exactly and tractably in the easier special case of the RBM, because $P(h|x)$ factorizes into $\prod_i P(h_i|x)$. Similarly $P(x|h)$ factorizes into $\prod_j P(x_j|h)$, which makes it possible to apply blocked Gibbs sampling (sampling h given x, then x given h, again h given x, etc.). For a trained RBM, the learned representation $R(x)$ of its input x is usually taken to be $E[h|x]$, as a heuristic.

RBMs are typically trained by stochastic gradient descent, using a noisy (and generally biased) estimator of the above log-likelihood gradient. The first gradient estimator that was proposed for RBMs is the Contrastive Divergence estimator (Hinton, 1999; Hinton *et al.*, 2006), and it has a particularly simple form: the negative phase gradient is obtained by starting a very short chain (usually just one step) at the observed x and replacing the above expectations by the corresponding samples. In practice, it has worked very well for unsupervised pre-training meant to initialize each layer of a deep supervised (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Erhan *et al.*, 2010) or unsupervised (Hinton and Salakhutdinov, 2006) neural network.

Another common way to train RBMs is based on the Stochastic Maximum Likelihood (SML) estimator (Younes, 1999) of the gradient, also called Persistent Contrastive Divergence (PCD; Tieleman, 2008) when it was introduced for RBMs. The idea is simply to keep sampling negative phase x's (e.g. by blocked Gibbs sampling) even though the parameters are updated once in a while, i.e., without restarting a new chain each time an update is done. It turned out that SML yields RBMs with much better likelihood, whereas CD updates sometimes give rise to worsening likelihood and suffer from other issues (Desjardins *et al.*, 2010). Theory suggests (Younes, 1999) this is a good estimator if the parameter changes are small, but practice revealed (Tieleman, 2008) that it worked even for large updates, in fact giving rise to faster mixing (Tieleman and Hinton, 2009; Breuleux *et al.*, 2011). This is happening because learning actually interacts with sampling in a useful way, pushing the MCMC out of the states it just visited. This principle may also explain some of the fast mixing observed in a related approach called Herding (Welling, 2009; Breuleux *et al.*, 2011).

RBMs can be stacked to form a Deep Belief Network (DBN), a hybrid of directed and undirected graphical model components, which has an RBM to characterize the interactions between its top two layers, and then generates the input through a directed belief network. See Bengio (2009) for a deeper treatment of Boltzmann Machines, RBMs, and Deep Belief Networks.

Auto-encoders are neural networks which are trained to reconstruct their input (Rumelhart *et al.*, 1986; Bourlard and Kamp, 1988; Hinton and Zemel, 1994). A one-hidden layer auto-encoder is very similar to an RBM and its reconstruction error gradient can be seen as an approximation of the RBM log-likelihood gradient (Bengio and Delalleau, 2009). Both RBMs and auto-encoders can be used as one-layer unsupervised learning algorithms that give rise to a new representation of the input or of the previous layer. In the same year that RBMs were successfully proposed for unsupervised pre-training of deep neural networks, auto-encoders were also shown to help initialize deep neural networks much better than random initialization (Bengio *et al.*, 2007). However, ordinary auto-encoders generally performed worse than RBMs, and were unsatisfying because they could potentially learn a useless identity transformation when the representation size was larger than the input (the so-called "overcomplete" case).

Sparse coding was introduced in computational neuroscience (Olshausen and Field, 1997) and produced filters very similar to those observed in cortex visual area V1 (before similar filters were achieved with RBMs, sparse predictive decomposition, and denoising auto-encoders, below). It corresponds to a linear directed graphical model with a continuous-valued latent variable associated with a sparsity prior (Student or Laplace, the latter corresponding to an L1 penalty on the value of the latent variable). This is like an auto-encoder, but without a parametric encoder, only a parametric decoder. The "encoding" corresponds to inference (finding the most likely hidden code associated with observed visible input) and involves solving a lengthy but convex optimization problem and much work has been devoted to speeding it up. A very interesting way to do so is with **Predictive Sparse Decomposition** (Kavukcuoglu *et al.*, 2008), in which one learns a parametric encoder that approximates the result of the sparse coding inference (and in fact changes the solution so that both approximate encoding and decoding work well). Such models based on approximate inference were the first successful examples of stacking a sparse encoding (Ranzato *et al.*, 2007; Jarrett *et al.*, 2009) into a deep architecture (fine-tuned for supervised classification afterwards, as per the above greedy-layerwise recipe).

Score Matching is an alternative statistical estimation principle (Hyvärinen, 2005) when the maximum likelihood framework is not tractable. It can be applied to models of continuous-valued data when the probability function can be computed tractably up to its normalization constant (which is the case for RBMs), i.e., it has a tractable energy function. The *score* of the model is the partial derivative of the log-likelihood with respect to the input, and indicates in which direction the likelihood would increase the most, from a particular input x. Score matching is based on minimizing the squared difference between the score of the model and a target score. The latter is in general unknown but the score match can nonetheless be rewritten in terms of the expectation (under the data generating process) of first and (diagonal) second derivatives of the energy with respect to the input, which correspond to a tractable computation.

Denoising Auto-Encoders were first introduced by Vincent *et al.* (2008) to bypass the frustrating limitations of auto-encoders mentioned above. Auto-encoders are only meant to learn a "bottleneck", a reduced-dimension representation. The idea of Denoising Auto-Encoders (DAE) is simple: feed the encoder/decoder system with a *stochastically corrupted input*, but ask it to *reconstruct the clean input* (as one would typically do to train any denoising system). This small change turned out to systematically yield better results than those obtained with ordinary auto-encoders, and similar or better than those obtained with RBMs on a benchmark of several image classification tasks (Vincent *et al.*, 2010). Interestingly, the denoising error can be linked in several ways to the likelihood of a generative model of the distribution of the uncorrupted examples (Vincent *et al.*, 2008; Vincent, 2011), and in particular through the Score Matching proxy for log-likelihood (Vincent, 2011): the denoising error corresponds to a form of *regularized* score matching criterion (Kingma and LeCun, 2010). The link also sheds light on why a denoising auto-encoder captures the input distribution. The difference vector between the reconstruction and the corrupted input is the model's guess as to the direction of greatest increase in the likelihood (starting from a corrupted example), whereas the difference vector between the corrupted input and the clean original is nature's hint of a direction of greatest increase in likelihood (since a noisy version of a training example is very likely to have a much lower probability than the original under the data generating distribution). The difference of these two differences is just the denoising reconstruction error residue.

Noise-Contrastive Estimation is another estimation principle which can be applied when the energy function can be computed but not the partition function (Gutmann and Hyvarinen, 2010). It is based on training not only from samples of the target distribution but also from samples of an auxiliary "background" distribution (e.g. a flat Gaussian). The partition function is considered like a free parameter (along with the other parameters) in a kind of logistic regression trained to predict the probability that a sample belongs to the target distribution vs the background distribution.

Semi-Supervised Embedding is an interesting and different way to use unlabeled data to learn a representation (e.g., in the hidden layers of a deep neural network), based on a hint about *pairs of examples* (Weston *et al.*, 2008). If two examples in a pair are expected to have a similar semantic, then their representations should be encouraged to be similar, whereas otherwise their representations should be at least some distance away. This idea was used in unsupervised and semi-supervised contexts (Chopra *et al.*, 2005; Hadsell *et al.*, 2006; Weston *et al.*, 2008), and originates in the much older idea of *siamese networks* (Bromley *et al.*, 1993).

Contractive autoencoders (Rifai *et al.*, 2011) minimize a training criterion that is the sum of a reconstruction error and a "contraction penalty", which encourages the learnt representation $h(x)$ to be as invariant as possible to the input x, while still allowing to distinguish the training examples from each other (i.e., to reconstruct them). As a consequence, the representation is faithful to changes in input space in the directions of the manifold near which examples concentrate, but it is highly contractive in the orthogonal directions. This is similar in spirit to a PCA (which only keeps the leading directions of variation and completely ignores the others), but is softer (no hard cutting at a particular dimension), is non-linear and can contract in different directions depending on where one looks in the input space (hence can capture non-linear manifolds). To prevent a trivial solution in which the encoder weights go to zero and the decoder weights to infinity, the contractive autoencoder uses tied weights (the decoder weights are forced to be the transpose of the encoder weights). Because of the contractive criterion, what we find empirically is that for any particular input example, many of the hidden units saturate while a few remain sensitive to changes in the input (corresponding to changes in the directions of changes expected under the data distribution). That subset of active units changes as we move around in input space, and defines a kind of *local chart*, or local coordinate system, in the neighborhood of each input point. This can be visualized to some extent by looking at the singular values and singular vectors of the Jacobian matrix J (containing the derivatives of each hidden unit output with respect to each input unit). Contrary to other autoencoders, one tends to find only few dominant eigenvalues, and their number corresponds to a local rank or local dimension (which can change as we move in input space). This is unlike other dimensionality reduction algorithms in which the number of dimensions is fixed by hand (rather than learnt) and fixed across the input domain. In fact the learnt representation can be overcomplete (larger than the input): it is only in the sense of its Jacobian that it has an effective small dimensionality for any particular input point. The large number of hidden units can be exploited to model complicated non-linear manifolds.

5 Principles, Challenges and Ideas Ahead

What lies beyond the principle of local generalization which has already been very successful in machine learning? The principles of distributed (possibly sparse) representations and deep circuits make possible other ways to generalize. Both can exploit a combinatorial effect in order to characterize and differentiate a number of input regions that is exponentially larger than the number of parameters.

However, these principles also come with challenges, notably a more difficult non-convex optimization problem. The greedy layer-wise unsupervised pre-training trick has served well, but more needs to be done in order to globally train these deep architectures. This optimization difficulty means that the optimization problem is not cleanly decoupled from the modeling choices. Some models may work well in practice because the optimization is easier.

Representation learning algorithms have been found empirically to partly disentangle the underlying factors of variation, such as geometric factors of variation (Goodfellow *et al.*, 2009), or domain vs sentiment in sentiment analysis (Glorot *et al.*, 2011). This means that some learned features (some component of the representation) are more invariant to some factors of variation (compared to the raw input) and more sensitive to others. Perhaps the most exciting challenge ahead is the following: why is this apparent disentangling happening, and can we go even further in that direction? We already know some tricks which seem to help this disentangling: independence (e.g., as in ICA), sparsity (e.g., as in sparse auto-encoders, sparse RBMs, or sparse denoising auto-encoders), grouping (forcing some groups of learned features to behave as a group, e.g., encouraging all of the units in a group to be off together), and slowness (forcing some units to respond in a temporally coherent manner). We propose to invest more towards understanding all of these better and exploiting this understanding to yield learning algorithms that better disentangle the underlying factors of variation in AI-related tasks.

Bibliography

Attwell, D. and Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow And Metabolism*, **21**, 1133–1145.

Barron, A. E. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. on Information Theory*, **39**, 930–945.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, **2**(1), 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y. (2011). Deep learning of representations for unsupervised and transfer learning. In *Workshop on Unsupervised and Transfer Learning (ICML'11)*.

Bengio, Y. and Delalleau, O. (2009). Justifying and generalizing contrastive divergence. *Neural Computation*, **21**(6), 1601–1621.

Bengio, Y. and Delalleau, O. (2011). Shallow versus deep sum-product networks. *The Learning Workshop*, Fort Lauderdale, Florida.

Bengio, Y. and LeCun, Y. (2007a). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press.

Bengio, Y. and LeCun, Y. (2007b). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press.

Bengio, Y. and Monperrus, M. (2005). Non-local manifold tangent learning. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pages 129–136. MIT Press.

Bengio, Y., Delalleau, O., and Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS'05)*, pages 107–114. MIT Press, Cambridge, MA.

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

Bengio, Y., Delalleau, O., and Simard, C. (2010). Decision trees do not generalize to new variations. *Computational Intelligence*, **26**(4), 449–467.

Bengio, Y., Bastien, F., Bergeron, A., Boulanger-Lewandowski, N., Breuel, T., Chherawala, Y., Cisse, M., Côté, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Pannetier Lebeuf, S., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

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

Braverman, M. (2011). Poly-logarithmic independence fools bounded-depth boolean circuits. *Communications of the ACM*, **54**(4), 108–115.

Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an rbm-derived process. *Neural Computation*, **23**(8), 2058–2073.

Bromley, J., Benz, J., Bottou, L., Guyon, I., Jackel, L., LeCun, Y., Moore, C., Sackinger, E., and Shah, R. (1993). Signature verification using a siamese time delay neural network. In *Advances in Pattern Recognition Systems using Neural Network Technologies*, pages 669–687. World Scientific, Singapore.

Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7 (NIPS'94)*, pages 657–664, Cambridge, MA. MIT Press.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'05)*. IEEE Press.

Collobert, R. and Weston, J. (2008a). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML 2008*.

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

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. (2010). Tempered Markov chain monte carlo for training of restricted Boltzmann machine. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 145–152.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, **11**, 625–660.

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

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*.

Goodfellow, I., Le, Q., Saxe, A., and Ng, A. (2009). Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 646–654.

Gutmann, M. and Hyvarinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*.

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

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

Håstad, J. (1986). Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th annual ACM Symposium on Theory of Computing*, pages 6–20, Berkeley, California. ACM Press.

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

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

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, **40**, 185–234.

Hinton, G. E. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)*, volume 1, pages 1–6, Edinburgh, Scotland. IEE.

Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and helmholtz free energy. In D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6 (NIPS'93)*, pages 3–10. Morgan Kaufmann Publishers, Inc.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). Boltzmann machines: Constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science.

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

Hyvärinen, A. (2005). Estimation of non-normalized statistical models using score matching. *Journal of Machine Learning Research*, **6**, 695–709.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pages 2146–2153. IEEE.

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

Kavukcuoglu, K., Ranzato, M., Fergus, R., and LeCun, Y. (2009). Learning invariant features through topographic filter maps. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'09)*, pages 1605–1612. IEEE.

Kingma, D. and LeCun, Y. (2010). Regularized estimation of image statistics by score matching. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1126–1134.

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

Larochelle, H., Erhan, D., and Vincent, P. (2009). Deep learning using robust interdependent codes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 312–319.

Le Roux, N. and Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, **20**(6), 1631–1649.

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

Lennie, P. (2003). The cost of cortical computation. *Current Biology*, **13**(6), 493–497.

Mesnil, G., Dauphin, Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I., Lavoie, E., Muller, X., Desjardins, G., Warde-Farley, D., Vincent, P., Courville, A., and Bergstra, J. (2011). Unsupervised and transfer learning challenge: a deep learning approach. In *Workshop on Unsupervised and Transfer Learning (ICML'11)*.

Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. In L. Bottou and M. Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 737–744, Montreal. Omnipress.

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

Osindero, S. and Hinton, G. E. (2008). Modeling image patches with a directed hierarchy of markov random field. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1121–1128, Cambridge, MA. MIT Press.

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Proceedings of the 27th Conference in Uncertainty in Artificial Intelligence (UAI)*, Barcelona, Spain.

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

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

Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*.

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

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

Salakhutdinov, R. and Hinton, G. E. (2009). Deep Boltzmann machines. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, volume 5, pages 448–455.

Salakhutdinov, R. and Hinton, G. E. (2010). An efficient learning procedure for deep Boltzmann machines. Technical Report MIT-CSAIL-TR-2010-037, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

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

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1064–1071. ACM.

Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, pages 1033–1040. ACM.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation*, **23**(7), 1661–1674.

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

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**(3371–3408).

Welling, M. (2009). Herding dynamic weights for partially observed random field models. In *Proceedings of the 25th Conference in Uncertainty in Artificial Intelligence (UAI'09)*. Morgan Kaufmann.

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

Wolpert, D. H. (1996). The lack of a priori distinction between learning algorithms. *Neural Computation*, **8**(7), 1341–1390.

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

Younes, L. (1999). On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics and Stochastic Reports*, **65**(3), 177–228.