Representation Learning: A Review and New Perspectives

Yoshua Bengio, Aaron Courville, and Pascal Vincent
Department of computer science and operations research, U. Montreal

Abstract—
The success of machine learning algorithms generally depends on data representation, and we hypothesize that this is because different representations can entangle and hide more or less the different explanatory factors of variation behind the data. Although specific domain knowledge can be used to help design representations, learning with generic priors can also be used, and the quest for AI is motivating the design of more powerful representation-learning algorithms implementing such priors. This paper reviews recent work in the area of unsupervised feature learning and joint training of deep learning, covering advances in probabilistic models, auto-encoders, manifold learning, and deep architectures. This motivates longer-term unanswered questions about the appropriate objectives for learning good representations, for computing representations (i.e., inference), and the geometrical connections between representation learning, density estimation and manifold learning.

Index Terms—Deep learning, representation learning, feature learning, unsupervised learning, Boltzmann Machine, RBM, auto-encoder, neural network

1 Introduction

The performance of machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied. For that reason, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations that result in a representation of the data that can support effective machine learning. Such feature engineering is important but labor-intensive and highlights the weakness of current learning algorithms: their inability to extract and organize the discriminative information from the data. Feature engineering is a way to take advantage of human ingenuity and prior knowledge to compensate for that weakness. In order to expand the scope and ease of applicability of machine learning, it would be highly desirable to make learning algorithms less dependent on feature engineering, so that novel applications could be constructed faster, and more importantly, to make progress towards Artificial Intelligence (AI). An AI must fundamentally *understand the world around us*, and we argue that this can only be achieved if it can learn to identify and disentangle the underlying explanatory factors hidden in the observed milieu of low-level sensory data.

This paper is about feature learning, or *representation learning*, i.e., learning transformations of the data that make it easier to extract useful information when building classifiers or other predictors. In the case of probabilistic models, a good representation is often one that captures the posterior distribution of the underlying explanatory factors for the observed input. Among the various ways of learning representations, this paper focuses on deep learning methods: those that are formed by the composition of multiple non-linear transformations of the data, with the goal of yielding more abstract – and ultimately more useful – representations. Here we survey this rapidly developing area with special emphasis on recent progress. We consider some of the fundamental questions that have been driving research in this area. Specifically, what makes one representation better than another? Given an example, how should we compute its representation, i.e. perform feature extraction? Also, what are appropriate objectives for learning good representations? In the course of dealing with these issues we review some of the most popular models in the field and place them in a context of the field as a whole.

2 Why should we care about learning representations?

Representation learning has become a field in itself in the machine learning community, with regular workshops at the leading conferences such as NIPS and ICML, sometimes under the header of *Deep Learning* or *Feature Learning*. Although depth is an important part of the story, many other priors are interesting and can be conveniently captured by a learner when the learning problem is cast as one of learning a representation, as discussed in the next section. The rapid increase in scientific activity on representation learning has been accompanied and nourished (in a virtuous circle) by a remarkable string of empirical successes both in academia and in industry. In this section, we briefly highlight some of these high points.

Speech Recognition and Signal Processing

Speech was one of the early applications of neural networks, in particular convolutional (or time-delay) neural networks [1]. The recent revival of interest in neural networks, deep learning, and representation learning has had a strong impact in the area of speech recognition, with breakthrough results (Dahl *et al.*, 2010; Seide *et al.*, 2011; Mohamed *et al.*, 2012; Dahl *et al.*, 2012) obtained by several academics as well as researchers at industrial labs taking over the task of bringing these algorithms to a larger scale and into products. For example, Microsoft has released in 2012 a new version of their MAVIS (Microsoft Audio Video Indexing Service) speech system based on deep learning (Seide *et al.*, 2011). These authors managed to reduce

1. See Bengio (1993) for a review of early work in this area.

the word error rate on four major benchmarks by about 30% (e.g. from 27.4% to 18.5% on RT03S) compared to state-of-the-art models based on Gaussian mixtures for the acoustic modeling and trained on the same amount of data (309 hours of speech). The relative improvement in error rate obtained by Dahl *et al.* (2012) on a smaller large-vocabulary speech recognition benchmark (Bing mobile business search dataset, with 40 hours of speech) is between 16% and 23%.

Representation-learning algorithms (based on recurrent neural networks) have also been applied to music, substantially beating the state-of-the-art in polyphonic transcription (Boulanger-Lewandowski *et al.*, 2012), with a relative error improvement of between 5% and 30% on a standard benchmark of four different datasets.

Object Recognition

The beginnings of deep learning in 2006 have focused on the MNIST digit image classification problem (Hinton *et al.*, 2006a; Bengio *et al.*, 2007), breaking the supremacy of SVMs (1.4% error) on this dataset[2]. The latest records are still held by deep networks: Ciresan *et al.* (2012) currently claims the title of state-of-the-art for the unconstrained version of the task (e.g., using a convolutional architecture), with 0.27% error, and Rifai *et al.* (2011c) is state-of-the-art for the knowledge-free version of MNIST, with 0.81% error.

In the last few years, deep learning has moved from digits to object recognition in natural images, and the latest breakthrough has been achieved on the ImageNet dataset[3] bringing down the state-of-the-art error rate from 26.1% to 15.3% (Krizhevsky *et al.*, 2012).

Natural Language Processing

Besides speech recognition, there are many other Natural Language Processing applications of representation learning algorithms. The idea of distributed representation for symbolic data was introduced by Hinton (1986), and first developed in the context of statistical language modeling by Bengio *et al.* (2003)[4]. They are all based on learning a distributed representation for each word, also called a *word embedding*. Combining this idea with a convolutional architecture, Collobert *et al.* (2011) developed the SENNA system[5] that shares representations across the tasks of language modeling, part-of-speech tagging, chunking, named entity recognition, semantic role labeling and syntactic parsing. SENNA approaches or surpasses the state-of-the-art on these tasks but is much faster than traditional predictors and requires only 3500 lines of C code to perform its predictions.

The neural net language model was also improved by adding recurrence to the hidden layers (Mikolov *et al.*, 2011), allowing it to beat the state-of-the-art (smoothed n-gram models) not only in terms of perplexity (exponential of the average negative log-likelihood of predicting the right next word, going down from 140 to 102) but also in terms of word error rate in speech recognition (since the language model is an important component of a speech recognition system), decreasing it from 17.2% (KN5 baseline) or 16.9% (discriminative language model) to 14.4% on the Wall Street Journal benchmark task. Similar models have been applied in statistical machine translation (Schwenk *et al.*, 2012), improving the BLEU score by almost 2 points. Recursive auto-encoders (which generalize recurrent networks) have also been used to beat the state-of-the-art in full sentence paraphrase detection (Socher *et al.*, 2011a) almost doubling the F1 score for paraphrase detection. Representation learning can also be used to perform word sense disambiguation (Bordes *et al.*, 2012), bringing up the accuracy from 67.8% to 70.2% on the subset of Senseval-3 where the system could be applied (with subject-verb-object sentences). Finally, it has also been successfully used to surpass the state-of-the-art in sentiment analysis (Glorot *et al.*, 2011b; Socher *et al.*, 2011b).

Multi-Task and Transfer Learning, Domain Adaptation

Transfer learning is the ability of a learning algorithm to exploit commonalities between different learning tasks in order to share statistical strength, and *transfer knowledge* across tasks. As discussed below, we hypothesize that representation learning algorithms have an advantage for such tasks because they learn representations that capture underlying factors, a subset of which may be relevant for each particular task, as illustrated in Figure 1. This hypothesis seems confirmed by a number of empirical results showing the strengths of representation learning algorithms in transfer learning scenarios.

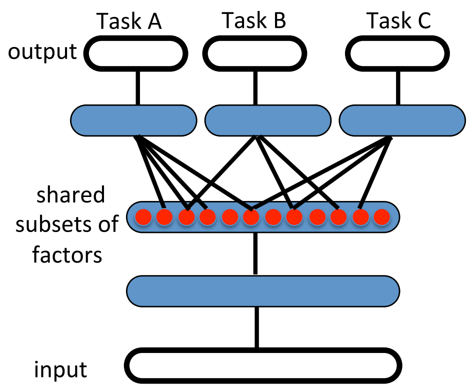

Fig. 1. Illustration of a representation-learning model which discovers explanatory factors (middle hidden layer, in red), some of which explain the input (semi-supervised setting), and some of which explain the target for each task. Because these subsets overlap, sharing of statistical strength allows gains in generalization.

Most impressive are the two transfer learning challenges held in 2011 and won by representation learning algorithms. First, the Transfer Learning Challenge, presented at an ICML 2011 workshop of the same name, was won using unsupervised layer-wise pre-training (Bengio, 2011; Mesnil *et al.*, 2011). A second Transfer Learning Challenge was held the same year and won by Goodfellow *et al.* (2011). Results were presented at NIPS 2011's Challenges in Learning Hierarchical Models Workshop. Other examples of the successful application of representation learning in fields related to transfer

2. for the knowledge-free version of the task, where no image-specific prior is used, such as image deformations or convolutions

3. The 1000-class ImageNet benchmark, whose results are detailed here: `http://www.image-net.org/challenges/LSVRC/2012/results.html`

4. See this review of *neural net language models* (Bengio, 2008).

5. downloadable from `http://ml.nec-labs.com/senna/`

learning include *domain adaptation*, where the target remains the same but the input distribution changes (Glorot *et al.*, 2011b; Chen *et al.*, 2012). Of course, the case of jointly predicting outputs for many tasks or classes, i.e., performing *multi-task* learning also enhances the advantage of representation learning algorithms, e.g. as in Krizhevsky *et al.* (2012); Collobert *et al.* (2011).

3 What makes a representation good?

3.1 Priors for Representation Learning in AI

In Bengio and LeCun (2007), one of us introduced the notion of AI-tasks, which are challenging for current machine learning algorithms, and involve complex but highly structured dependencies. One reason why explicitly dealing with representations is interesting is because they can be convenient to express many general priors about the world around us, i.e., priors that are not task-specific but would be likely to be useful for a learning machine to solve AI-tasks. Examples of such general-purpose priors are the following:

- **Smoothness**: we want to learn functions f s.t. $x \approx y$ generally implies $f(x) \approx f(y)$. This is the most basic prior and is present in most machine learning, but is insufficient to get around the curse of dimensionality, as discussed in Section 3.2 below.
- **Multiple explanatory factors**: the data generating distribution is generated by different underlying factors, and for the most part what one learns about one factor generalizes in many configurations of the other factors. The objective to recover or at least disentangle these underlying factors of variation is discussed in Section 3.5. This assumption is behind the idea of **distributed representations**, discussed in Section 3.3 below.
- **A hierarchical organization of explanatory factors**: the concepts that are useful at describing the world around us can be defined in terms of other concepts, in a hierarchy, with more **abstract** concepts higher in the hierarchy, being defined in terms of less abstract ones. This is the assumption exploited by having **deep representations**, elaborated in Section 3.4 below.
- **Semi-supervised learning**: in the context where we have input variables X and target variables Y we may want to predict, a subset of the factors that explain X's distribution explain a great deal of Y, given X. Hence representations that are useful for $P(X)$ tend to be useful when learning $P(Y|X)$, allowing sharing of statistical strength between the unsupervised and supervised learning tasks, as discussed in Section 4.
- **Shared factors across tasks**: in the context where we have many Y's of interest or many learning tasks in general, tasks (e.g., the corresponding $P(Y|X, \text{task})$) are explained by factors that are shared with other tasks, allowing sharing of statistical strengths across tasks, as discussed in the previous section (Multi-Task and Transfer Learning, Domain Adaptation).
- **Manifolds**: probability mass concentrates near regions that have a much smaller dimensionality than the original space where the data lives. This is explicitly exploited in some of the auto-encoder algorithms and other manifold-inspired algorithms described respectively in Sections 7.2 and 8.
- **Natural clustering**: different values of categorical variables such as object classes[6] are associated with separate manifolds. More precisely, the local variations on the manifold tend to preserve the value of a category, and a linear interpolation between examples of different classes in general involves going through a low density region, i.e., $P(X|Y = i)$ for different i tend to be well separated and not overlap much. For example, this is exploited in the Manifold Tangent Classifier discussed in Section 8.3. This hypothesis is consistent with the idea that humans have *named* categories and classes because of such statistical structure (discovered by their brain and propagated by their culture), and machine learning tasks often involves predicting such categorical variables.
- **Temporal and spatial coherence**: this is similar to the cluster assumption but concerns sequences of observations; consecutive or spatially nearby observations tend to be associated with the same value of relevant categorical concepts, or result in a small move on the surface of the high-density manifold. More generally, different factors change at different temporal and spatial scales, and many categorical concepts of interest change slowly. When attempting to capture such categorical variables, this prior can be enforced by making the associated representations slowly changing, i.e., penalizing changes in values over time or space. This prior was introduced in Becker and Hinton (1992) and is discussed in Section 11.3.
- **Sparsity**: for any given observation x, only a small fraction of the possible factors are relevant. In terms of representation, this could be represented by features that are often zero (as initially proposed by Olshausen and Field (1996)), or by the fact that most of the extracted features are *insensitive* to small variations of x. This can be achieved with certain forms of priors on latent variables (peaked at 0), or by using a non-linearity whose value is often flat at 0 (i.e., 0 and with a 0 derivative), or simply by penalizing the magnitude of the Jacobian matrix (of derivatives) of the function mapping input to representation. This is discussed in Sections 6.1.3 and 7.2.

We can view many of the above priors as ways to help the learner discover and **disentangle** some of the underlying (and a priori unknown) factors of variation that the data may reveal. This idea is pursued further in Sections 3.5 and 11.4.

3.2 Smoothness and the Curse of Dimensionality

For AI-tasks, such as computer vision and natural language understanding, it seems hopeless to rely only on simple parametric models (such as linear models) because they cannot capture enough of the complexity of interest. Conversely, machine learning researchers have sought flexibility in *local*[7] *non-parametric* learners such as kernel machines with

6. it is often the case that the Y of interest is a category

7. *local* in the sense that the value of the learned function at x depends mostly on training examples $x^{(t)}$'s close to x

a fixed generic local-response kernel (such as the Gaussian kernel). Unfortunately, as argued at length by Bengio and Monperrus (2005); Bengio *et al.* (2006a); Bengio and LeCun (2007); Bengio (2009); Bengio *et al.* (2010), most of these algorithms only exploit the principle of *local generalization*, i.e., the assumption that the target function (to be learned) is smooth enough, so they rely on examples to *explicitly map out the wrinkles of the target function*. Generalization is mostly achieved by a form of local interpolation between neighboring training examples. Although smoothness can be a useful assumption, it is insufficient to deal with the *curse of dimensionality*, because the number of such wrinkles (ups and downs of the target function) may grow exponentially with the number of relevant interacting factors, when the data are represented in raw input space. We advocate learning algorithms that are flexible and non-parametric[8] but do not rely exclusively on the smoothness assumption. Instead, we propose to incorporate generic priors such as those enumerated above into representation-learning algorithms. Smoothness-based learners (such as kernel machines) and linear models can still be useful on top of such learned representations. In fact, the combination of learning a representation and kernel machine is equivalent to *learning the kernel*, i.e., the feature space. Kernel machines are useful, but they depend on a prior definition of a suitable similarity metric, or a feature space in which naive similarity metrics suffice. We would like to use the data, along with very generic priors, to discover those features, or equivalently, a similarity function.

3.3 Distributed representations

Good representations are *expressive*, meaning that a reasonably-sized learned representation can capture a huge number of possible input configurations. A simple counting argument helps us to assess the expressiveness of a model producing a representation: how many parameters does it require compared to the number of input regions (or configurations) it can distinguish? A one-hot representations, such as the result of traditional clustering algorithms, a Gaussian mixture model, a nearest-neighbor algorithm, a decision tree, or a Gaussian SVM all require $O(N)$ parameters (and/or $O(N)$ examples) to distinguish $O(N)$ input regions. One could naively believe that in order to define $O(N)$ input regions one cannot do better. However, RBMs, sparse coding, auto-encoders or multi-layer neural networks can all represent up to $O(2^k)$ input regions using only $O(N)$ parameters (with k the number of non-zero elements in a sparse representation, and $k = N$ in non-sparse RBMs and other dense representations). These are all *distributed representations* (where k elements can independently be varied, e.g., they are not mutually exclusive) or sparse (distributed representations where only a few of the elements can be varied at a time). The generalization of clustering to distributed representations is *multi-clustering*, where either several clusterings take place in parallel or the same clustering is applied on different parts of the input, such as in the very popular hierarchical feature extraction for object recognition based on a histogram of cluster categories detected in different patches of an image (Lazebnik *et al.*, 2006; Coates and Ng, 2011a). The exponential gain from distributed or sparse representations is discussed further in section 3.2 (and Figure 3.2) of Bengio (2009). It comes about because each parameter (e.g. the parameters of one of the units in a sparse code, or one of the units in a Restricted Boltzmann Machine) can be re-used in many examples that are not simply near neighbors of each other, whereas with local generalization, different regions in input space are basically associated with their own private set of parameters, e.g., as in decision trees, nearest-neighbors, Gaussian SVMs, etc. In a distributed representation, an exponentially large number of possible *subsets* of features or hidden units can be activated in response to a given input. In a single-layer model, each feature is typically associated with a preferred input direction, corresponding to a hyperplane in input space, and the *code* or representation associated with that input is precisely the pattern of activation (which features respond to the input, and how much). This is in contrast with a non-distributed representation such as the one learned by most clustering algorithms, e.g., k-means, in which the representation of a given input vector is a one-hot code identifying which one of a small number of cluster centroids best represents the input [9].

3.4 Depth and abstraction

Depth is a key aspect to representation learning strategies we consider in this paper. As we will discuss, deep architectures are often challenging to train effectively and this has been the subject of much recent research and progress. However, despite these challenges, they carry two significant advantages that motivate our long-term interest in discovering successful training strategies for deep architectures. These advantages are: (1) deep architectures promote the *re-use* of features, and (2) deep architectures can potentially lead to progressively more *abstract* features at higher layers of representations (more removed from the data).

Feature re-use. The notion of re-use, which explains the power of distributed representations, is also at the heart of the theoretical advantages behind *deep learning*, i.e., constructing multiple levels of representation or learning a hierarchy of features. The depth of a circuit is the length of the longest path from an input node of the circuit to an output node of the circuit. The crucial property of a deep circuit is that its number of paths, i.e., *ways to re-use different parts*, can grow exponentially with its depth. Formally, one can change the depth of a given circuit by changing the definition of what

8. We understand *non-parametric* as including all learning algorithms whose capacity can be increased appropriately as the amount of data and its complexity demands it, e.g. including mixture models and neural networks where the number of parameters is a data-selected hyper-parameter.

9. As discussed in (Bengio, 2009), things are only slightly better when allowing continuous-valued membership values, e.g., in ordinary mixture models (with separate parameters for each mixture component), but the difference in representational power is still exponential (Montufar and Morton, 2012). The situation may also seem better with a decision tree, where each given input is associated with a one-hot code over the tree leaves, which deterministically selects associated ancestors (the path from root to node). Unfortunately, the number of different regions represented (equal to the number of leaves of the tree) still only grows linearly with the number of parameters used to specify it (Bengio and Delalleau, 2011).

each node can compute, but only by a constant factor. The typical computations we allow in each node include: weighted sum, product, artificial neuron model (such as a monotone non-linearity on top of an affine transformation), computation of a kernel, or logic gates. Theoretical results clearly show families of functions where a deep representation can be exponentially more efficient than one that is insufficiently deep (Håstad, 1986; Håstad and Goldmann, 1991; Bengio *et al.*, 2006a; Bengio and LeCun, 2007; Bengio and Delalleau, 2011). If the same family of functions can be represented with fewer parameters (or more precisely with a smaller VC-dimension, learning theory would suggest that it can be learned with fewer examples, yielding improvements in both *computational* efficiency (less nodes to visit) and *statistical* efficiency (less parameters to learn, and re-use of these parameters over many different kinds of inputs).

Abstraction and invariance. Deep architectures can lead to abstract representations because more abstract concepts can often be constructed in terms of less abstract ones. In some cases, such as in the convolutional neural network (LeCun *et al.*, 1998b), we build this abstraction in explicitly via a pooling mechanism (see section 11.2). More abstract concepts are generally *invariant* to most local changes of the input. That makes the representations that capture these concepts generally highly non-linear functions of the raw input. This is obviously true of categorical concepts, where more abstract representations detect categories that cover more varied phenomena (e.g. larger manifolds with more wrinkles) and thus they potentially have greater predictive power. Abstraction can also appear in high-level continuous-valued attributes that are only sensitive to some very specific types of changes in the input. Learning these sorts of invariant features has been a long-standing goal in pattern recognition.

3.5 Disentangling Factors of Variation

Beyond being *distributed* and *invariant*, we would like our representations to *disentangle the factors of variation*. Different explanatory factors of the data tend to change independently of each other in the input distribution, and only a few at a time tend to change when one considers a sequence of consecutive real-world inputs.

Complex data arise from the rich interaction of many sources. These factors interact in a complex web that can complicate AI-related tasks such as object classification. For example, an image is composed of the interaction between one or more light sources, the object shapes and the material properties of the various surfaces present in the image. Shadows from objects in the scene can fall on each other in complex patterns, creating the illusion of object boundaries where there are none and dramatically effect the perceived object shape. How can we cope with these complex interactions? How can we *disentangle* the objects and their shadows? Ultimately, we believe the approach we adopt for overcoming these challenges must leverage the data itself, using vast quantities of unlabeled examples, to learn representations that separate the various explanatory sources. Doing so should give rise to a representation significantly more robust to the complex and richly structured variations extant in natural data sources for AI-related tasks.

It is important to distinguish between the related but distinct goals of learning invariant features and learning to disentangle explanatory factors. The central difference is the preservation of information. Invariant features, by definition, have reduced sensitivity in the direction of invariance. This is the goal of building features that are insensitive to variation in the data that are uninformative to the task at hand. Unfortunately, it is often difficult to determine *a priori* which set of features will ultimately be relevant to the task at hand. Further, as is often the case in the context of deep learning methods, the feature set being trained may be destined to be used in multiple tasks that may have distinct subsets of relevant features. Considerations such as these lead us to the conclusion that the most robust approach to feature learning is *to disentangle as many factors as possible, discarding as little information about the data as is practical*. If some form of dimensionality reduction is desirable, then we hypothesize that the local directions of variation least represented in the training data should be first to be pruned out (as in PCA, for example, which does it globally instead of around each example).

3.6 What are good criteria for learning representations?

One of the challenges of representation learning that distinguishes it from other machine learning tasks such as classification is the difficulty in establishing a clear objective, or target for training. In the case of classification, the objective is (at least conceptually) obvious, we want to minimize the number of misclassifications on the training dataset. In the case of representation learning, our objective is far-removed from the ultimate objective, which is typically learning a classifier or some other predictor. Our problem is reminiscent of the credit assignment problem encountered in reinforcement learning. We have proposed that a good representation is one that disentangles the underlying factors of variation, but how do we translate that into appropriate training criteria? Is it even necessary to do anything but maximize likelihood under a good model or can we introduce priors such as those enumerated above (possibly *data-dependent* ones) that help the representation better do this disentangling? This question remains clearly open but is discussed in more detail in Sections 3.5 and 11.4.

4 Building Deep Representations

In 2006, a breakthrough in feature learning and deep learning was initiated by Geoff Hinton and quickly followed up in the same year (Hinton *et al.*, 2006a; Bengio *et al.*, 2007; Ranzato *et al.*, 2007). It has been extensively reviewed and discussed in Bengio (2009). A central idea, referred to as *greedy layerwise unsupervised pre-training*, was to learn a hierarchy of features one level at a time, using unsupervised feature learning to learn a new transformation at each level to be composed with the previously learned transformations; essentially, each iteration of unsupervised feature learning adds one layer of weights to a deep neural network. Finally, the set of layers could be combined to initialize a deep supervised predictor, such as a neural network classifier, or a deep generative model, such as a Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009).

This paper is mostly about feature learning algorithms that can be used to form deep architectures. In particular, it was empirically observed that layerwise *stacking* of feature extraction often yielded better representations, e.g., in terms of classification error (Larochelle *et al.*, 2009; Erhan *et al.*, 2010b), quality of the samples generated by a probabilistic model (Salakhutdinov and Hinton, 2009) or in terms of the *invariance* properties of the learned features (Goodfellow *et al.*, 2009). Whereas this section focuses on the idea of stacking single-layer models, Section 10 follows up with a discussion on joint training of all the layers.

The greedy layerwise unsupervised pre-training procedure (Hinton *et al.*, 2006a; Bengio *et al.*, 2007; Bengio, 2009) is based on training each layer with an unsupervised representation learning algorithm, taking the features produced at the previous level as input for the next level. It is then straightforward to use the resulting deep feature extraction either as input to a standard supervised machine learning predictor (such as an SVM) or as initialization for a deep supervised neural network (e.g., by appending a logistic regression layer or purely supervised layers of a multi-layer neural network). The layerwise procedure can also be applied in a purely *supervised* setting, called the *greedy layerwise supervised pre-training* (Bengio *et al.*, 2007). For example, after the first one-hidden-layer MLP is trained, its output layer is discarded and another one-hidden-layer MLP can be stacked on top of it, etc. Although results reported in Bengio *et al.* (2007) were not as good as for unsupervised pre-training, they were nonetheless better than without pre-training at all. Alternatively, the *outputs* of the previous layer can be fed as *extra inputs* for the next layer, as successfully done in Yu *et al.* (2010).

Whereas combining single layers into a supervised model is straightforward, it is less clear how layers pre-trained by unsupervised learning should be combined to form a better *unsupervised* model. We cover here some of the approaches to do so, but no clear winner emerges and much work has to be done to validate existing proposals or improve them.

The first proposal was to stack pre-trained RBMs into a Deep Belief Network (Hinton *et al.*, 2006a) or DBN, where the top layer is interpreted as an RBM and the lower layers as a directed sigmoid belief network. However, it is not clear how to approximate maximum likelihood training to further optimize this generative model. One option is the wake-sleep algorithm (Hinton *et al.*, 2006a) but more work should be done to assess the efficiency of this procedure in terms of improving the generative model.

The second approach that has been put forward is to combine the RBM parameters into a Deep Boltzmann Machine (DBM), by basically halving the RBM weights to obtain the DBM weights (Salakhutdinov and Hinton, 2009). The DBM can then be trained by approximate maximum likelihood as discussed in more details later (Section 10.2). This joint training has brought substantial improvements, both in terms of likelihood and in terms of classification performance of the resulting deep feature learner (Salakhutdinov and Hinton, 2009).

Another early approach was to stack RBMs or auto-encoders into a *deep auto-encoder* (Hinton and Salakhutdinov, 2006). If we have a series of encoder-decoder pairs $(f^{(i)}(\cdot), g^{(i)}(\cdot))$, then the overall encoder is the composition of the encoders, $f^{(N)}(\dots f^{(2)}(f^{(1)}(\cdot)))$, and the overall decoder is its "transpose" (often with transposed weight matrices as well), $g^{(1)}(g^{(2)}(\dots f^{(N)}(\cdot)))$. The deep auto-encoder (or its regularized version, as discussed in Section 7.2) can then be jointly trained, with all the parameters optimized with respect to a common training criterion. More work on this avenue clearly needs to be done, and it was probably avoided by fear of the challenges in training deep feedforward networks, discussed in the Section 10 along with very encouraging recent results.

Yet another recently proposed approach to training deep architectures (Ngiam *et al.*, 2011) is to consider the iterative construction of a *free energy function* (i.e., with no explicit latent variables, except possibly for a top-level layer of hidden units) for a deep architecture as the composition of transformations associated with lower layers, followed by top-level hidden units. The question is then how to train a model defined by an arbitrary parametrized (free) energy function. Ngiam *et al.* (2011) have used Hybrid Monte Carlo (Neal, 1993), but other options include contrastive divergence (Hinton *et al.*, 2006b), score matching (Hyvärinen, 2005a; Hyvärinen, 2008), denoising score matching (Kingma and LeCun, 2010; Vincent, 2011), and noise-contrastive estimation (Gutmann and Hyvarinen, 2010).

5 Single-layer learning modules

Within the community of researchers interested in representation learning, there has developed two broad parallel lines of inquiry: one rooted in probabilistic graphical models and one rooted in neural networks. Fundamentally, the difference between these two paradigms is whether the layered architecture of a deep learning model is to be interpreted as describing a probabilistic graphical model or as describing a computation graph. In short, are hidden units considered latent random variables or as computational nodes?

To date, the dichotomy between these two paradigms has remained in the background, perhaps because they appear to have more characteristics in common than separating them. We suggest that this is likely a function of the fact that much recent progress in both of these areas has focused on *single-layer greedy learning modules* and the similarities between the types of single-layer models that have been explored: mainly, the restricted Boltzmann machine (RBM) on the probabilistic side, and the auto-encoder variants on the neural network side. Indeed, as shown by one of us (Vincent, 2011) and others (Swersky *et al.*, 2011), in the case of the restricted Boltzmann machine, training the model via an inductive principle known as score matching (Hyvärinen, 2005b) (to be discussed in sec. 6.4.3) is essentially identical to a regularized reconstruction objective of an auto-encoder. Another strong link between pairs of models on both sides of this divide is when the computational graph for computing representation in the neural network model corresponds exactly to the computational graph that corresponds to inference in the probabilistic model, and this happens to also correspond to the structure of graphical model itself.

The connection between these two paradigms becomes more tenuous when we consider deeper models where, in the case of a probabilistic model, exact inference typically becomes intractable. In the case of deep models, the computational graph diverges from the structure of the model. For example, in the case of a deep Boltzmann machine, unrolling variational (approximate) inference into a computational graph results in a recurrent graph structure. We have performed preliminary exploration (Savard, 2011) of deterministic variants of deep auto-encoders whose computational graph is similar to that of a deep Boltzmann machine (in fact very close to the mean-field variational approximations associated with the Boltzmann machine), and that is one interesting intermediate point to explore (between the deterministic approaches and the graphical model approaches).

In the next few sections we will review the major developments in single-layer training modules used to support feature learning and particularly deep learning. We divide these sections between (Section 6) the probabilistic models, with inference and training schemes that directly parametrize the generative – or *decoding* – pathway and (Section 7) the typically neural network-based models that directly parametrize the *encoding* pathway. Interestingly, some models, like Predictive Sparse Decomposition (PSD) (Kavukcuoglu *et al.*, 2008) inherit both properties, and will also be discussed (Section 7.2.4). We then present a different view of representation learning, based on the associated geometry and the manifold assumption, in Section 8.

Before we do this, we consider an unsupervised single-layer representation learning algorithm that spans all three views (probabilistic, auto-encoder, and manifold learning) discussed here.

Principal Components Analysis

We will use probably the oldest feature extraction algorithm, principal components analysis (PCA) (Pearson, 1901; Hotelling, 1933), to illustrate the probabilistic, auto-encoder and manifold views of representation-learning. PCA learns a linear transformation $h = f(x) = W^T x + b$ of input $x \in \mathbb{R}^{d_x}$, where the columns of $d_x \times d_h$ matrix W form an orthogonal basis for the d_h orthogonal directions of greatest variance in the training data. The result is d_h features (the components of representation h) that are decorrelated. The three interpretations of PCA are the following: a) it is related to *probabilistic models* (Section 6) such as probabilistic PCA, factor analysis and the traditional multivariate Gaussian distribution (the leading eigenvectors of the covariance matrix are the principal components); b) the representation it learns is essentially the same as that learned by a basic linear *auto-encoder* (Section 7.2); and c) it can be viewed as a simple linear form of linear *manifold learning* (Section 8), i.e., characterizing a lower-dimensional region in input space near which the data density is peaked. Thus, PCA may be in the back of the reader's mind as a common thread relating these various viewpoints. Unfortunately the expressive power of linear features is very limited: they cannot be stacked to form deeper, more abstract representations since the composition of linear operations yields another linear operation. Here, we focus on recent algorithms that have been developed to extract *non-linear* features, which can be stacked in the construction of deep networks, although some authors simply insert a non-linearity between learned single-layer linear projections (Le *et al.*, 2011c; Chen *et al.*, 2012).

Another rich family of feature extraction techniques that this review does not cover in any detail due to space constraints is Independent Component Analysis or ICA (Jutten and Herault, 1991; Comon, 1994; Bell and Sejnowski, 1997). Instead, we refer the reader to Hyvärinen *et al.* (2001a); Hyvärinen *et al.* (2009). Note that, while in the simplest case (complete, noise-free) ICA yields linear features, in the more general case it can be equated with a *linear generative model* with non-Gaussian independent latent variables, similar to sparse coding (section 6.1.3), which result in *non-linear features*. Therefore, ICA and its variants like Independent and Topographic ICA (Hyvärinen *et al.*, 2001b) can and have been used to build deep networks (Le *et al.*, 2010, 2011c): see section 11.2. The notion of obtaining independent components also appears similar to our stated goal of disentangling underlying explanatory factors through deep networks. However, for complex real-world distributions, it is doubtful that the relationship between truly independent underlying factors and the observed high-dimensional data can be adequately characterized by a linear transformation.

6 Probabilistic Models

From the probabilistic modeling perspective, the question of feature learning can be interpreted as an attempt to recover a parsimonious set of latent random variables that describe a distribution over the observed data. We can express any probabilistic model over the joint space of the latent variables, h, and observed or visible variables x, (associated with the data) as $p(x, h)$. Feature values are conceived as the result of an inference process to determine the probability distribution of the latent variables given the data, i.e. $p(h \mid x)$, often referred to as the *posterior* probability. Learning is conceived in term of estimating a set of model parameters that (locally) maximizes the likelihood of the training data with respect to the *distribution over these latent variables*. The probabilistic graphical model formalism gives us two possible modeling paradigms in which we can consider the question of inferring latent variables: directed and undirected graphical models. The key distinguishing factor between these paradigms is the nature of their parametrization of the joint distribution $p(x, h)$. The choice of directed versus undirected model has a major impact on the nature and computational costs of the algorithmic approach to both inference and learning.

6.1 Directed Graphical Models

Directed latent factor models are parametrized through a decomposition of the joint distribution, $p(x, h) = p(x \mid h)p(h)$, involving a *prior* $p(h)$, and a likelihood $p(x \mid h)$ that describes the observed data x in terms of the latent factors h. Unsupervised feature learning models that can be interpreted with this decomposition include: Principal Components Analysis (PCA) (Roweis, 1997; Tipping and Bishop, 1999), sparse coding (Olshausen and Field, 1996), sigmoid belief networks (Neal, 1992) and the newly introduced spike-and-slab sparse coding model (Goodfellow *et al.*, 2011).

6.1.1 *Explaining Away*

In the context of latent factor models, the form of the directed model often leads to one important property, namely explaining away: *a priori* independent causes of an event can become non-independent given the observation of the event. Latent factor models can generally be interpreted as latent *cause* models, where the h activations cause the observed x. This renders the *a priori* independent h to be non-independent. As a consequence, recovering the posterior distribution of h, $p(h \mid x)$ (which we use as a basis for feature representation), is often computationally challenging and can be entirely intractable, especially when h is discrete.

A classic example that illustrates the phenomenon is to imagine you are on vacation away from home and you receive a phone call from the company that installed the security system at your house. They tell you that the alarm has been activated. You begin worrying your home has been burglarized, but then you hear on the radio that a minor earthquake has been reported in the area of your home. If you happen to know from prior experience that earthquakes sometimes cause your home alarm system to activate, then suddenly you relax, confident that your home has very likely not been burglarized.

The example illustrates how the observation, **alarm activation**, rendered two otherwise entirely independent causes, **burglarized** and **earthquake**, to become dependent – in this case, the dependency is one of mutual exclusivity. Since both **burglarized** and **earthquake** are very rare events and both can cause **alarm activation**, the observation of one *explains away* the other. The example demonstrates not only how observations can render causes to be statistically dependent, but also the utility of explaining away. It gives rise to a parsimonious prediction of the unseen or latent events from the observations. Returning to latent factor models, despite the computational obstacles we face when attempting to recover the posterior over h, explaining away promises to provide a parsimonious $p(h \mid x)$, which can be an extremely useful characteristic of a feature encoding scheme. If one thinks of a representation as being composed of various feature detectors and estimated attributes of the observed input, it is useful to allow the different features to compete and collaborate with each other to explain the input. This is naturally achieved with directed graphical models, but can also be achieved with undirected models (see Section 6.2) such as Boltzmann machines if there are *lateral connections* between the corresponding units or corresponding *interaction terms* in the energy function that defines the probability model.

6.1.2 *Probabilistic Interpretation of PCA*

While PCA was not originally cast as probabilistic model, it possesses a natural probabilistic interpretation (Roweis, 1997; Tipping and Bishop, 1999) that casts PCA as *factor analysis*:

$$\begin{aligned} p(h) &= \mathcal{N}(h; 0, \sigma_h^2 \mathbf{I}) \\ p(x \mid h) &= \mathcal{N}(x; Wh + \mu_x, \sigma_x^2 \mathbf{I}), \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^{d_x}$, $h \in \mathbb{R}^{d_h}$, $\mathcal{N}(v; \mu, \Sigma)$ is the multivariate normal density of v with mean μ and covariance Σ, and columns of W span the same space as leading d_h principal components, but are not constrained to be orthonormal.

6.1.3 *Sparse Coding*

As in the case of PCA, sparse coding has both a probabilistic and non-probabilistic interpretation. Sparse coding also relates a latent representation h (either a vector of random variables or a feature vector, depending on the interpretation) to the data x through a linear mapping W, which we refer to as the dictionary. The difference between sparse coding and PCA is that sparse coding includes a penalty to ensure a sparse activation of h is used to encode each input x.

Specifically, from a non-probabilistic perspective, sparse coding can be seen as recovering the code or feature vector associated with a new input x via:

$$h^* = f(x) = \underset{h}{\operatorname{argmin}} \|x - Wh\|_2^2 + \lambda \|h\|_1, \tag{2}$$

Learning the dictionary W can be accomplished by optimizing the following training criterion with respect to W:

$$\mathcal{J}_{\text{SC}} = \sum_t \|x^{(t)} - Wh^{*(t)}\|_2^2, \tag{3}$$

where the $x^{(t)}$ is the input vector for example t and $h^{*(t)}$ are the corresponding sparse codes determined by Eq. 2. W is usually constrained to have unit-norm columns (because one can arbitrarily exchange scaling of column i with scaling of $h_i^{(t)}$, such a constraint is necessary for the L1 penalty to have any effect).

The probabilistic interpretation of sparse coding differs from that of PCA, in that instead of a Gaussian prior on the latent random variable h, we use a sparsity inducing Laplace prior (corresponding to an L1 penalty):

$$\begin{aligned} p(h) &= \prod_i^{d_h} \lambda \exp(-\lambda |h_i|) \\ p(x \mid h) &= \mathcal{N}(x; Wh + \mu_x, \sigma_x^2 \mathbf{I}). \end{aligned} \tag{4}$$

In the case of sparse coding, because we will seek a sparse representation (i.e., one with many features set to exactly zero), we will be interested in recovering the MAP (maximum *a posteriori* value of h: i.e. $h^* = \operatorname{argmax}_h p(h \mid x)$ rather than its expected value $\mathbb{E}_{[} [h] \,|x]$. Under this interpretation, dictionary learning proceeds as maximizing the likelihood of the data *given these MAP values of* h^*: $\operatorname{argmax}_W \prod_t p(x^{(t)} \mid h^{*(t)})$ subject to the norm constraint on W. Note that this parameter learning scheme, subject to the MAP values of the latent h, is not standard practice in the probabilistic graphical model literature. Typically the likelihood of the data $p(x) = \sum_h p(x \mid h) p(h)$ is maximized directly. In the presence of latent variables, expectation maximization (Dempster *et al.*, 1977) is employed where the parameters are optimized with respect to the marginal likelihood, i.e., summing or integrating the joint log-likelihood over the values of the latent variables under their posterior $P(h \mid x)$, rather than considering only the MAP values of h. The theoretical properties of this form of parameter learning are not yet well understood but seem to work well in practice (e.g. k-Means vs Gaussian mixture models and Viterbi training for HMMs). Note also that the interpretation of sparse coding as a MAP estimation can be questioned (Gribonval, 2011), because even though the interpretation of the L1 penalty as a log-prior is a possible interpretation, there can be other Bayesian interpretations compatible with the training criterion.

Sparse coding is an excellent example of the power of explaining away. The Laplace distribution (equivalently, the L1 penalty) over the latent h acts to resolve a sparse and parsimonious representation of the input. Even with a very overcomplete dictionary with many redundant bases, the MAP inference process used in sparse coding to find h^* can pick out the most appropriate bases and zero the others, despite them having a high degree of correlation with the input. This property arises naturally in directed graphical models such as sparse coding and is entirely owing to the explaining away effect. It is not seen in commonly used undirected probabilistic models such as the RBM, nor is it seen in parametric feature encoding methods such as auto-encoders. The trade-off is that, compared to methods such as RBMs and auto-encoders, inference in sparse coding involves an extra inner-loop of optimization to find h^* with a corresponding increase in the computational cost of feature extraction. Compared to auto-encoders and RBMs, the code in sparse coding is a free variable for each example, and in that sense the implicit encoder is non-parametric.

One might expect that the parsimony of the sparse coding representation and its explaining away effect would be advantageous and indeed it seems to be the case. Coates and Ng (2011a) demonstrated with the CIFAR-10 object classification task (Krizhevsky and Hinton, 2009) with a patch-base feature extraction pipeline, that in the regime with few ($<$ 1000) labeled training examples per class, the sparse coding representation significantly outperformed other highly competitive encoding schemes. Possibly because of these properties, and because of the very computationally efficient algorithms that have been proposed for it (in comparison with the general case of inference in the presence of explaining away), sparse coding enjoys considerable popularity as a feature learning and encoding paradigm. There are numerous examples of its successful application as a feature representation scheme, including natural image modeling (Raina *et al.*, 2007; Kavukcuoglu *et al.*, 2008; Coates and Ng, 2011a; Yu *et al.*, 2011), audio classification (Grosse *et al.*, 2007), natural language processing (Bagnell and Bradley, 2009), as well as being a very successful model of the early visual cortex (Olshausen and Field, 1997). Sparsity criteria can also be generalized successfully to yield groups of features that prefer to all be zero, but if one or a few of them are active then the penalty for activating others in the group is small. Different *group sparsity* patterns can incorporate different forms of prior knowledge (Kavukcuoglu *et al.*, 2009; Jenatton *et al.*, 2009; Bach *et al.*, 2011; Gregor *et al.*, 2011).

Spike-and-Slab Sparse Coding. Spike-and-slab sparse coding (S3C) is one example of a promising variation on sparse coding for feature learning (Goodfellow *et al.*, 2012). The S3C model possesses a set of latent binary *spike* variables together with a a set of latent real-valued *slab* variables. The activation of the spike variables dictate the sparsity pattern. S3C has been applied to the CIFAR-10 and CIFAR-100 object classification tasks (Krizhevsky and Hinton, 2009), and shows the same pattern as sparse coding of superior performance in the regime of relatively few ($<$ 1000) labeled examples per class (Goodfellow *et al.*, 2012). In fact, in both the CIFAR-100 dataset (with 500 examples per class) and the CIFAR-10 dataset (when the number of examples is reduced to a similar range), the S3C representation actually outperforms sparse coding representations. This advantage was revealed clearly with S3C winning the NIPS'2011 Transfer Learning Challenge (Goodfellow *et al.*, 2011).

6.2 Undirected Graphical Models

Undirected graphical models, also called Markov random fields (MRFs), parametrize the joint $p(x, h)$ through a factorization in terms of unnormalized non-negative *clique potentials*:

$$p(x,h) = \frac{1}{Z_\theta} \prod_i \psi_i(x) \prod_j \eta_j(h) \prod_k \nu_k(x,h) \tag{5}$$

where $\psi_i(x)$, $\eta_j(h)$ and $\nu_k(x, h)$ are the clique potentials describing the interactions between the visible elements, between the hidden variables, and those interaction between the visible and hidden variables respectively. The partition function Z_θ ensures that the distribution is normalized. Within the context of unsupervised feature learning, we generally see a particular form of Markov random field called a Boltzmann distribution with clique potentials constrained to be positive:

$$p(x,h) = \frac{1}{Z_\theta} \exp\left(-\mathcal{E}_\theta(x,h)\right), \tag{6}$$

where $\mathcal{E}_\theta(x, h)$ is the energy function and contains the interactions described by the MRF clique potentials and θ are the model parameters that characterize these interactions.

A Boltzmann machine is defined as a network of symmetrically-coupled binary random variables or units. These stochastic units can be divided into two groups: (1) the *visible* units $x \in \{0,1\}^{d_x}$ that represent the data, and (2) the *hidden* or latent units $h \in \{0,1\}^{d_h}$ that mediate dependencies between the visible units through their mutual interactions. The pattern of interaction is specified through the energy function:

$$\mathcal{E}_\theta^{\mathrm{BM}}(x,h) = -\frac{1}{2}x^T U x - \frac{1}{2}h^T V h - x^T W h - b^T x - d^T h, \tag{7}$$

where $\theta = \{U, V, W, b, d\}$ are the model parameters which respectively encode the visible-to-visible interactions, the hidden-to-hidden interactions, the visible-to-hidden interactions, the visible self-connections, and the hidden self-connections (also known as biases). To avoid over-parametrization, the diagonals of U and V are set to zero.

The Boltzmann machine energy function specifies the probability distribution over the joint space $[x, h]$, via the Boltzmann distribution, Eq. 6, with the partition function Z_θ given by:

$$Z_\theta = \sum_{x_1=0}^{x_1=1} \cdots \sum_{x_{d_x}=0}^{x_{d_x}=1} \sum_{h_1=0}^{h_1=1} \cdots \sum_{h_{d_h}=0}^{h_{d_h}=1} \exp\left(-\mathcal{E}_\theta^{\mathrm{BM}}(x,h;\theta)\right). \tag{8}$$

This joint probability distribution gives rise to the set of conditional distributions of the form:

$$P(h_i \mid x, h_{\setminus i}) = \mathrm{sigmoid}\left(\sum_j W_{ji}x_j + \sum_{i' \neq i} V_{ii'}h_{i'} + d_i\right) \quad (9)$$

$$P(x_j \mid h, x_{\setminus j}) = \mathrm{sigmoid}\left(\sum_i W_{ji}x_j + \sum_{j' \neq j} U_{jj'}x_{j'} + b_j\right). \quad (10)$$

In general, inference in the Boltzmann machine is intractable. For example, computing the conditional probability of h_i given the visibles, $P(h_i \mid x)$, requires marginalizing over the rest of the hiddens, which implies evaluating a sum with 2^{d_h-1} terms:

$$P(h_i \mid x) = \sum_{h_1=0}^{h_1=1} \cdots \sum_{h_{i-1}=0}^{h_{i-1}=1} \sum_{h_{i+1}=0}^{h_{i+1}=1} \cdots \sum_{h_{d_h}=0}^{h_{d_h}=1} P(h \mid x) \quad (11)$$

However with some judicious choices in the pattern of interactions between the visible and hidden units, more tractable subsets of the model family are possible, as we discuss next.

6.2.1 Restricted Boltzmann Machines

The restricted Boltzmann machine (RBM) is likely the most popular subclass of Boltzmann machine (Smolensky, 1986). It is defined by restricting the interactions in the Boltzmann energy function, in Eq. 7, to only those between h and x, i.e. $\mathcal{E}_\theta^{\mathrm{RBM}}$ is $\mathcal{E}_\theta^{\mathrm{BM}}$ with $U = \mathbf{0}$ and $V = \mathbf{0}$. As such, the RBM can be said to form a *bipartite* graph with the visibles and the hiddens forming two layers of vertices in the graph (and no connection between units of the same layer). With this restriction, the RBM possesses the useful property that the conditional distribution over the hidden units factorizes given the visibles:

$$P(h \mid x) = \prod_i P(h_i \mid x)$$
$$P(h_i = 1 \mid x) = \mathrm{sigmoid}\left(\sum_j W_{ji}x_j + d_i\right) \quad (12)$$

Likewise, the conditional distribution over the visible units given the hiddens also factorizes:

$$P(x \mid h) = \prod_j P(x_j \mid h)$$
$$P(x_j = 1 \mid h) = \mathrm{sigmoid}\left(\sum_i W_{ji}h_i + b_j\right) \quad (13)$$

This conditional factorization property of the RBM immediately implies that most inferences we would like to make are readily tractable. For example, the RBM feature representation is taken to be the set of posterior marginals $P(h_i \mid x)$, which, given the conditional independence described in Eq. 12, are immediately available. Note that this is in stark contrast to the situation with popular directed graphical models for unsupervised feature extraction, where computing the posterior probability is intractable.

Importantly, the tractability of the RBM does not extend to its partition function, which still involves summing an exponential number of terms. It does imply however that we can limit the number of terms to $\min\{2^{d_x}, 2^{d_h}\}$. Usually this is still an unmanageable number of terms and therefore we must resort to approximate methods to deal with its estimation.

It is difficult to overstate the impact the RBM has had to the fields of unsupervised feature learning and deep learning. It has been used in a truly impressive variety of applications, including fMRI image classification (Schmah *et al.*, 2009), motion and spatial transformations (Taylor and Hinton, 2009; Memisevic and Hinton, 2010), collaborative filtering (Salakhutdinov *et al.*, 2007) and natural image modeling (Ranzato and Hinton, 2010; Courville *et al.*, 2011b).

6.3 Generalizations of the RBM to Real-valued data

Important progress has been made in the last few years in defining generalizations of the RBM that better capture real-valued data, in particular real-valued image data, by better modeling the conditional covariance of the input pixels. The standard RBM, as discussed above, is defined with both binary visible variables $v \in \{0, 1\}$ and binary latent variables $h \in \{0, 1\}$. The tractability of inference and learning in the RBM has inspired many authors to extend it, via modifications of its energy function, to model other kinds of data distributions. In particular, there has been multiple attempts to develop RBM-type models of real-valued data, where $x \in \mathbb{R}^{d_x}$. The most straightforward approach to modeling real-valued observations within the RBM framework is the so-called *Gaussian RBM* (GRBM) where the only change in the RBM energy function is to the visible units biases, by adding a bias term that is quadratic in the visible units x. While it probably remains the most popular way to model real-valued data within the RBM framework, Ranzato and Hinton (2010) suggest that the GRBM has proved to be a somewhat unsatisfactory model of natural images. The trained features typically do not represent sharp edges that occur at object boundaries and lead to latent representations that are not particularly useful features for classification tasks. Ranzato and Hinton (2010) argue that the failure of the GRBM to adequately capture the statistical structure of natural images stems from the exclusive use of the model capacity to capture the conditional mean at the expense of the conditional covariance. Natural images, they argue, are chiefly characterized by the covariance of the pixel values, not by their absolute values. This point is supported by the common use of preprocessing methods that standardize the global scaling of the pixel values across images in a dataset or across the pixel values within each image.

These kinds of concerns about the ability of the GRBM to model natural image data has lead to the development of alternative RBM-based models that each attempt to take on this objective of *better modeling non-diagonal conditional covariances*. (Ranzato and Hinton, 2010) introduced the *mean and covariance RBM* (mcRBM). Like the GRBM, the mcRBM is a 2-layer Boltzmann machine that explicitly models the visible units as Gaussian distributed quantities. However unlike the GRBM, the mcRBM uses its hidden layer to independently parametrize both the mean and covariance of the data through two sets of hidden units. The mcRBM is a combination of the covariance RBM (cRBM) (Ranzato *et al.*, 2010a), that models the conditional covariance, with the GRBM that captures the conditional mean. While the GRBM has shown considerable potential as the basis of a highly successful phoneme recognition system (Dahl *et al.*, 2010), it seems that due to difficulties in training the mcRBM, the model has been largely superseded by the mPoT model. The mPoT model (*mean-product of Student's T-distributions model*) (Ranzato *et al.*, 2010b) is a combination of the GRBM and the product of Student's T-distributions model (Welling *et al.*, 2003). It is an energy-based model where the conditional distribution over the visible units conditioned on the hidden variables is a multivariate Gaussian (non-diagonal covariance) and the complementary conditional distribution over the hidden variables given the visibles are a set of independent Gamma distributions.

The PoT model has recently been generalized to the mPoT model (Ranzato *et al.*, 2010b) to include nonzero Gaussian means by the addition of GRBM-like hidden units, similarly to how the mcRBM generalizes the cRBM. The mPoT model has been used to synthesize large-scale natural images (Ranzato *et al.*, 2010b) that show large-scale features and shadowing structure. It has been used to model natural textures (Kivinen and Williams, 2012) in a *tiled-convolution* configuration (see section 11.2).

Another recently introduced RBM-based model with the objective of having the hidden units encode both the mean and covariance information is the *spike-and-slab* Restricted Boltzmann Machine (ssRBM) (Courville *et al.*, 2011a,b). The ssRBM is defined as having both a real-valued "slab" variable and a binary "spike" variable associated with each unit in the hidden layer. The ssRBM has been demonstrated as a feature learning and extraction scheme in the context of CIFAR-10 object classification (Krizhevsky and Hinton, 2009) from natural images and has performed well in the role (Courville *et al.*, 2011a,b). When trained *convolutionally* (see Section 11.2) on full CIFAR-10 natural images, the model demonstrated the ability to generate natural image samples that seem to capture the broad statistical structure of natural images better than previous parametric generative models, as illustrated with the samples of Figure 2.

The mcRBM, mPoT and ssRBM each set out to model real-valued data such that the hidden units encode not only the conditional mean of the data but also its conditional covariance. Other than differences in the training schemes, the most significant difference between these models is how they encode their conditional covariance. While the mcRBM and the mPoT use the activation of the hidden units to enforce constraints on the covariance of x, the ssRBM uses the hidden unit to pinch the precision matrix along the direction specified by the corresponding weight vector. These two ways of modeling conditional covariance diverge when the dimensionality of the hidden layer is significantly different from that of the input. In the over-complete setting, sparse activation with the ssRBM parametrization permits variance only in the select directions of the sparsely activated hidden units. This is a property the ssRBM shares with sparse coding models (Olshausen and Field, 1997; Grosse *et al.*, 2007). On the other hand, in the case of the mPoT or mcRBM, an over-complete set of constraints on the covariance implies that capturing arbitrary covariance along a particular direction of the input requires decreasing potentially all constraints with positive projection in that direction. This perspective would suggest that the mPoT and mcRBM do not appear to be well suited to provide a sparse representation in the overcomplete setting.

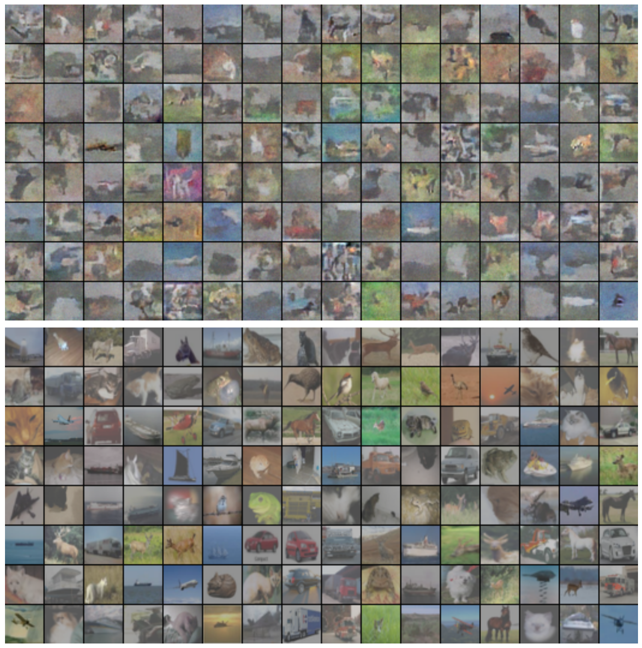
Fig. 2. (Top) Samples from a convolutionally trained μ-ssRBM, see details in Courville *et al.* (2011b). (Bottom) The images in the CIFAR-10 training set closest (L2 distance with contrast normalized training images) to the *corresponding* model samples. The model does not appear to be capturing the natural image statistical structure by overfitting particular examples from the dataset.

6.4 RBM parameter estimation

In this section we discuss several algorithms for training the restricted Boltzmann machine. Many of the methods we discuss are applicable to more general undirected graphical models, but are particularly practical in the RBM setting. Freund and Haussler (1994) proposed a learning algorithm for harmoniums (RBMs) based on projection pursuit (Friedman and Stuetzle, 1981). Contrastive Divergence (Hinton, 1999; Hinton *et al.*, 2006a) has been used most often to train RBMs, and many recent papers use Stochastic Maximum Likelihood (Younes, 1999; Tieleman, 2008).

As discussed in Sec. 6.1, in training probabilistic models parameters are typically adapted in order to maximize the *likelihood of the training data* (or equivalently the log-likelihood, or its penalized version, which adds a regularization term). With T training examples, the log likelihood is given by:

$$\sum_{t=1}^{T} \log P(x^{(t)}; \theta) = \sum_{t=1}^{T} \log \sum_{h \in \{0,1\}^{d_h}} P(x^{(t)}, h; \theta). \qquad (14)$$

One straightforward way we can consider maximizing this quantity is to take small steps uphill, following the log-likelihood gradient, to find a local maximum of the likelihood. For any Boltzmann machine, the gradient of the log-likelihood of the data is given by:

$$\frac{\partial}{\partial \theta_i} \sum_{t=1}^{T} \log p(x^{(t)}) = -\sum_{t=1}^{T} \mathbb{E}_{p(h|x^{(t)})} \left[\frac{\partial}{\partial \theta_i} \mathcal{E}_{\theta}^{\mathrm{BM}}(x^{(t)}, h) \right] + \sum_{t=1}^{T} \mathbb{E}_{p(x,h)} \left[\frac{\partial}{\partial \theta_i} \mathcal{E}_{\theta}^{\mathrm{BM}}(x, h) \right], \quad (15)$$

where we have the expectations with respect to $p(h^{(t)} \mid x^{(t)})$ in the "clamped" condition (also called the positive phase), and over the full joint $p(x, h)$ in the "unclamped" condition (also called the negative phase). Intuitively, the gradient acts to locally move the model distribution (the negative phase distribution) toward the data distribution (positive phase distribution), by pushing down the energy of $(h, x^{(t)})$ pairs (for $h \sim P(h|x^{(t)})$) while pushing up the energy of (h, x) pairs (for $(h, x) \sim P(h, x)$) until the two forces are in equilibrium, at which point the sufficient statistics (gradient of the energy function) have equal expectations with x sampled from the training distribution or with x sampled from the model.

The RBM conditional independence properties imply that the expectation in the positive phase of Eq. 15 is readily tractable. The negative phase term – arising from the partition function's contribution to the log-likelihood gradient – is more problematic because the computation of the expectation over the joint is not tractable. The various ways of dealing with the partition function's contribution to the gradient have brought about a number of different training algorithms, many trying to approximate the log-likelihood gradient.

To approximate the expectation of the joint distribution in the negative phase contribution to the gradient, it is natural to again consider exploiting the conditional independence of the RBM in order to specify a Monte Carlo approximation of the expectation over the joint:

$$\mathbb{E}_{p(x,h)} \left[\frac{\partial}{\partial \theta_i} \mathcal{E}_{\theta}^{\mathrm{RBM}}(x, h) \right] \approx \frac{1}{L} \sum_{l=1}^{L} \frac{\partial}{\partial \theta_i} \mathcal{E}_{\theta}^{\mathrm{RBM}}(\tilde{x}^{(l)}, \tilde{h}^{(l)}), \quad (16)$$

with the samples $(\tilde{x}^{(l)}, \tilde{h}^{(l)})$ drawn by a block Gibbs MCMC (Markov chain Monte Carlo) sampling scheme from the model distribution:

$$\begin{aligned} \tilde{x}^{(l)} &\sim P(x \mid \tilde{h}^{(l-1)}) \\ \tilde{h}^{(l)} &\sim P(h \mid \tilde{x}^{(l)}). \end{aligned}$$

Naively, for each gradient update step, one would start a Gibbs sampling chain, wait until the chain converges to the equilibrium distribution and then draw a sufficient number of samples to approximate the expected gradient with respect to the model (joint) distribution in Eq. 16. Then restart the process for the next step of approximate gradient ascent on the log-likelihood. This procedure has the obvious flaw that waiting for the Gibbs chain to "burn-in" and reach equilibrium anew for each gradient update cannot form the basis of a practical training algorithm. Contrastive Divergence (Hinton, 1999; Hinton *et al.*, 2006a), Stochastic Maximum Likelihood (Younes, 1999; Tieleman, 2008) and fast-weights persistent contrastive divergence or FPCD (Tieleman and Hinton, 2009) are all examples of algorithms that attempt sidestep the need to burn-in the negative phase Markov chain.

6.4.1 Contrastive Divergence:

Contrastive divergence (CD) estimation (Hinton, 1999; Hinton *et al.*, 2006a) uses a biased estimate of the gradient in Eq. 15 by approximating the negative phase expectation with a very short Gibbs chain (often just one step) initialized *at the training data used in the positive phase*. This initialization is chosen to reduce the variance of the negative expectation based on samples from the short running Gibbs sampler. The intuition is that, while the samples drawn from very short Gibbs chains may be a heavily biased (and poor) representation of the model distribution, they are at least moving in the direction of the model distribution relative to the data distribution represented by the positive phase training data. Consequently, they may combine to produce a good estimate of the gradient, or direction of progress. Much has been written about the properties and alternative interpretations of CD, e.g. Carreira-Perpiñan and Hinton (2005); Yuille (2005); Bengio and Delalleau (2009); Sutskever and Tieleman (2010).

6.4.2 Stochastic Maximum Likelihood:

The Stochastic Maximum Likelihood (SML) algorithm (also known as persistent contrastive divergence or PCD) (Younes, 1999; Tieleman, 2008) is an alternative way to sidestep an extended burn-in of the negative phase Gibbs sampler. At each gradient update, rather than initializing the Gibbs chain at the positive phase sample as in CD, SML initializes the chain at the last state of the chain used for the previous update. In other words, SML uses a continually running Gibbs chain (or often a number of Gibbs chains run in parallel) from which samples are drawn to estimate the negative phase expectation. Despite the model parameters changing between updates, these changes should be small enough that only a few steps of Gibbs (in practice, often one step is used) are required to maintain samples from the equilibrium distribution of the Gibbs chain, i.e. the model distribution.

One aspect of SML that has received considerable recent attention is that it relies on the Gibbs chain to have reasonably good mixing properties for learning to succeed. Typically, as learning progresses and the weights of the RBM grow, the ergodicity of the Gibbs sample begins to break down[10]. If the learning rate ϵ associated with gradient ascent $\theta \leftarrow \theta + \epsilon \hat{g}$ (with $E[\hat{g}] \approx \frac{\partial \log p_{\theta}(x)}{\partial \theta}$) is not reduced to compensate, then the Gibbs sampler will diverge from the model distribution and learning will fail. There have been a number of attempts made to address the failure of Gibbs chain mixing in the context of SML. Desjardins *et al.* (2010); Cho *et al.* (2010); Salakhutdinov (2010b,a) have all considered various forms of tempered transitions to improve the mixing rate of the negative phase Gibbs chain.

Tieleman and Hinton (2009) have proposed quite a different approach to addressing potential mixing problems of SML with their fast-weights persistent contrastive divergence

10. When weights become large, the estimated distribution is more peaky, and the chain takes very long time to mix, to move from mode to mode, so that practically the gradient estimator can be very poor. This is a serious chicken-and-egg problem because if sampling is not effective, nor is the training procedure, which may seem to stall.

(FPCD), and it has also been exploited to train Deep Boltzmann Machines (Salakhutdinov, 2010a) and construct a pure sampling algorithm for RBMs (Breuleux *et al.*, 2011). FPCD builds on the surprising but robust tendency of Gibbs chains to mix better during SML learning than when the model parameters are fixed. The phenomenon is rooted in the form of the likelihood gradient itself (Eq. 15). The samples drawn from the SML Gibbs chain are used in the negative phase of the gradient, which implies that the learning update will slightly increase the energy (decrease the probability) of those samples, making the region in the neighborhood of those samples less likely to be resampled and therefore making it more likely that the samples will move somewhere else (typically going near another mode). Rather than drawing samples from the distribution of the current model (with parameters θ), FPCD exaggerates this effect by drawing samples from a local perturbation of the model with parameters θ^* and an update specified by:

$$\theta^*_{t+1} = (1-\eta)\theta_{t+1} + \eta\theta^*_t + \epsilon^* \frac{\partial}{\partial \theta_i}\left(\sum_{t=1}^{T} \log p(x^{(t)})\right), \tag{17}$$

where ϵ^* is the relatively large fast-weight learning rate ($\epsilon^* > \epsilon$) and $0 < \eta < 1$ (but near 1) is a forgetting factor that keeps the perturbed model close to the current model. Unlike tempering, FPCD does not converge to the model distribution as ϵ and ϵ^* go to 0, and further work is necessary to characterize the nature of its approximation to the model distribution. Nevertheless, FPCD is a popular and apparently effective means of drawing approximate samples from the model distribution that faithfully represent its diversity, at the price of sometimes generating spurious samples *in between two modes* (because the fast weights roughly correspond to a smoothed view of the current model's energy function). It has been applied in a variety of applications (Tieleman and Hinton, 2009; Ranzato *et al.*, 2011; Kivinen and Williams, 2012) and it has been transformed into a sampling algorithm (Breuleux *et al.*, 2011) that also shares this fast mixing property with *herding* (Welling, 2009), for the same reason, i.e., introducing *negative correlations* between consecutive samples of the chain in order to promote faster mixing.

6.4.3 Pseudolikelihood, Ratio-matching and other Inductive Principles

While CD, SML and FPCD are by far the most popular methods for training RBMs and RBM-based models, all of these methods are perhaps most naturally described as offering different approximations to maximum likelihood training. There exist other inductive principles that are alternatives to maximum likelihood that can also be used to train RBMs. In particular, these include pseudo-likelihood (Besag, 1975) and ratio-matching (Hyvärinen, 2007). Both of these inductive principles attempt to avoid explicitly dealing with the partition function, and their asymptotic efficiency has been analyzed (Marlin and de Freitas, 2011). Pseudo-likelihood seeks to maximize the product of all one-dimensional conditional distributions of the form $P(x_d|x_{\setminus d})$, while ratio-matching can be interpreted as an extension of score matching (Hyvärinen, 2005a) to discrete data types. Both methods amount to weighted differences of the gradient of the RBM free energy[11] evaluated at a data point and at all neighboring points within a hamming ball of radius 1. One drawback of these methods is that the computation of the statistics for all neighbors of each training data point require a significant computational overhead, scaling linearly with the dimensionality of the input, n_d. CD, SML and FPCD have no such issue. Marlin *et al.* (2010) provides an excellent survey of these methods and their relation to CD and SML. They also empirically compared all of these methods on a range of classification, reconstruction and density modeling tasks and found that, in general, SML provided the best combination of overall performance and computational tractability. However, in a later study, the same authors (Swersky *et al.*, 2011) found *denoising score matching* (Kingma and LeCun, 2010; Vincent, 2011) to be a competitive inductive principle both in terms of classification performance (with respect to SML) and in terms of computational efficiency (with respect to analytically obtained score matching). Note that denoising score matching is a special case of the denoising auto-encoder training criterion (Section 7.2.2) when the reconstruction error residual equals a gradient, i.e., the score function associated with an energy function, as shown in (Vincent, 2011).

In the spirit of the Boltzmann machine update rule (Eq. 15) several other principles have been proposed to train energy-based models. One approach is *noise-contrastive estimation* (Gutmann and Hyvarinen, 2010), in which the training criterion is transformed into a *probabilistic classification problem*: distinguish between (positive) training examples and (negative) noise samples generated by a broad distribution (such as the Gaussian). Another family of approaches, more in the spirit of Contrastive Divergence, relies on distinguishing positive examples (of the training distribution) and negative examples obtained by slight perturbations of the positive examples (Collobert and Weston, 2008; Bordes *et al.*, 2012; Weston *et al.*, 2010). This apparently simple principle has been used successfully to train a model on huge quantities of data to map images and queries in the same space for Google's image search (Weston *et al.*, 2010).

7 Direct Encoding: Learning a Parametric Map from Input to Representation

Within the framework of probabilistic models adopted in Section 6, the learned representation is always associated with latent variables, specifically with their posterior distribution given an observed input x. Unfortunately, the posterior distribution of latent variables given inputs tends to become very complicated and intractable if the model has more than a couple of interconnected layers, whether in the directed or undirected graphical model frameworks. It then becomes necessary to resort to sampling or approximate inference techniques, and to pay the associated computational and approximation error price. This is in addition to the difficulties raised by the intractable partition function in undirected graphical

11. The free energy $\mathcal{F}(x;\theta)$ is defined in relation to the marginal likelihood of the data: $\mathcal{F}(x;\theta) = -\log P(x) - \log Z_\theta$ and in the case of the RBM is tractable.

models. Moreover a posterior *distribution* over latent variables is not yet a simple usable *feature vector* that can for example be fed to a classifier. So actual feature values are typically *derived* from that distribution, taking the latent variable's expectation (as is typically done with RBMs), their marginal probability, or finding their most likely value (as in sparse coding). If we are to extract stable deterministic numerical feature values in the end anyway, an alternative (apparently) non-probabilistic feature learning paradigm that focuses on carrying out this part of the computation, very efficiently, is that of auto-encoders and other directly parametrized feature or representation functions. The commonality between these methods is that they *learn a direct encoding, i.e., parametric map from inputs to their representation*,

The regularized auto-encoders are described in the next section, and are concerned with the case where the encoding function that computes the representation is associated with a decoding function that maps back to input space. In sections 8.1 and 11.3, we consider some direct encoding methods that do not require a decoder and a reconstruction error, such as semi-supervised embedding (Weston *et al.*, 2008) and slow feature analysis (Wiskott and Sejnowski, 2002).

7.1 Auto-Encoders

Whereas probabilistic models sometimes define intermediate variables whose posterior can then be interpreted as a representation, in the auto-encoder framework (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1994), one starts by explicitly defining a feature-extracting function in a specific parametrized closed form. This function, that we will denote f_θ, is called the **encoder** and will allow the straightforward and efficient computation of a feature vector $h = f_\theta(x)$ from an input x. For each example $x^{(t)}$ from a data set $\{x^{(1)}, \ldots, x^{(T)}\}$, we define

$$h^{(t)} = f_\theta(x^{(t)}) \tag{18}$$

where $h^{(t)}$ is the *feature-vector* or *representation* or *code* computed from $x^{(t)}$. Another closed form parametrized function g_θ, called the **decoder**, maps from feature space back into input space, producing a **reconstruction** $r = g_\theta(h)$. Whereas probabilistic models are defined from an explicit probability function and are trained to maximize (often approximately) the data likelihood (or a proxy), auto-encoders are parametrized through their encoder and decoder and are trained using a different training principle. The set of parameters θ of the encoder and decoder are learned simultaneously on the task of reconstructing as well as possible the original input, i.e. attempting to incur the lowest possible **reconstruction error** $L(x, r)$ – a measure of the discrepancy between x and its reconstruction – on average over a training set. Note how the main objective is to make reconstruction error low *on the training examples*, and by generalization, where the probability is high under the unknown data-generating distribution. For the minimization of reconstruction error to capture the structure of the data-generating distribution, it is therefore important that something in the training criterion or the parametrization prevents the auto-encoder from learning the identity function, which would yield zero reconstruction error everywhere. This is achieved through various means in the different forms of auto-encoders, as described below in more detail, and we call these *regularized auto-encoders*. A particular form of regularization consists in constraining the code to have a low dimension, and this is what the classical auto-encoder or PCA do.

In summary, basic auto-encoder training consists in finding a value of parameter vector θ minimizing reconstruction error

$$\mathcal{J}_{\text{DAE}}(\theta) \;=\; \sum_t L(x^{(t)}, g_\theta(f_\theta(x^{(t)}))) \tag{19}$$

where $x^{(t)}$ is a training example. This minimization is usually carried out by stochastic gradient descent as in the training of Multi-Layer-Perceptrons (MLPs). Since auto-encoders were primarily developed as MLPs predicting their input, the most commonly used forms for the encoder and decoder are affine mappings, optionally followed by a non-linearity:

$$f_\theta(x) \;=\; s_f(b + Wx) \tag{20}$$
$$g_\theta(h) \;=\; s_g(d + W'h) \tag{21}$$

where s_f and s_g are the encoder and decoder activation functions (typically the element-wise sigmoid or hyperbolic tangent non-linearity, or the identity function if staying linear). The set of parameters of such a model is $\theta = \{W, b, W', d\}$ where b and d are called encoder and decoder bias vectors, and W and W' are the encoder and decoder weight matrices.

The choice of s_g and L depends largely on the input domain range. and nature, and are usually chosen so that L returns a negative log-likelihood for the observed value of x. A natural choice for an unbounded domain is a linear decoder with a squared reconstruction error, i.e. $s_g(a) = a$ and $L(x, r) = \|x - r\|^2$. If inputs are bounded between 0 and 1 however, ensuring a similarly-bounded reconstruction can be achieved by using $s_g = \text{sigmoid}$. In addition if the inputs are of a binary nature, a binary cross-entropy loss[12] is sometimes used.

In the case of a linear auto-encoder (linear encoder and decoder) with squared reconstruction error, the basic auto-encoder objective in Equation 19 is known to learn the same *subspace*[13] as PCA. This is also true when using a sigmoid nonlinearity in the encoder (Bourlard and Kamp, 1988), but not if the weights W and W' are tied ($W' = W^T$).

Similarly, Le *et al.* (2011b) recently showed that adding a regularization term of the form $\sum_i \sum_j s_3(W_j x_i)$ to a linear auto-encoder with tied weights, where s_3 is a nonlinear convex function, yields an efficient algorithm for learning *linear ICA*.

If both encoder and decoder use a sigmoid non-linearity, then $f_\theta(x)$ and $g_\theta(h)$ have the exact *same form* as the conditionals $P(h \mid v)$ and $P(v \mid h)$ of binary RBMs (see Section 6.2.1). This similarity motivated an initial study (Bengio *et al.*, 2007) of the possibility of replacing RBMs with auto-encoders as the basic pre-training strategy for building deep networks, as well as the comparative analysis of auto-encoder reconstruction error gradient and contrastive divergence updates (Bengio and Delalleau, 2009).

12. $L(x, r) = -\sum_{i=1}^{d_x} x_i \log(r_i) + (1 - r_i) \log(1 - r_i)$

13. Contrary to traditional PCA loading factors, but similarly to the parameters learned by probabilistic PCA, the weight vectors learned by such an auto-encoder are not constrained to form an orthonormal basis, nor to have a meaningful ordering. They will however span the same subspace.

One notable difference in the parametrization is that RBMs use a single weight matrix, which follows naturally from their energy function, whereas the auto-encoder framework allows for a different matrix in the encoder and decoder. In practice however, *weight-tying* in which one defines $W' = W^T$ may be (and is most often) used, rendering the parametrizations identical. The usual training procedures however differ greatly between the two approaches. A practical advantage of training auto-encoder variants is that **they define a simple tractable optimization objective that can be used to monitor progress**.

7.2 Regularized Auto-Encoders

Traditionally, auto-encoders, like PCA, were primarily seen as a dimensionality reduction technique and thus used a *bottleneck*, i.e. $d_h < d_x$. But successful uses of sparse coding and RBM approaches tend to favour learning *over-complete* representations, i.e. $d_h > d_x$. This can render the auto-encoding problem too simple (e.g. simply duplicating the input in the features may allow perfect reconstruction without having extracted more meaningful features). Thus alternative ways to "constrain" the representation, other than constraining its dimensionality, have been investigated. We broadly refer to these alternatives as "regularized" auto-encoders. The effect of a bottleneck or of these regularization terms is that the auto-encoder cannot reconstruct well everything, it is trained to reconstruct well the training examples and generalization means that reconstruction error is also small on test examples. An interesting justification (Ranzato *et al.*, 2008) for the sparsity penalty (or any penalty that restricts in a soft way the volume of hidden configurations easily accessible by the learner) is that it acts in spirit like the partition function of RBMs, by making sure that only few input configurations can have a low reconstruction error.

Alternatively, one can view the objective of the regularization applied to an auto-encoder is to make the representation as "constant" (insensitive) as possible with respect to changes in input. This view immediately justifies two variants of regularized auto-encoders described below: contractive auto-encoders reduce the number of effective degrees of freedom of the representation (around each point) by making the encoder contractive, i.e., making the derivative of the encoder small (thus making the hidden units saturate), while the denoising auto-encoder makes the whole mapping "robust", i.e., insensitive to small random perturbations, or contractive, making sure that the reconstruction cannot be good when moving in most directions around a training example.

7.2.1 *Sparse Auto-Encoders*

The earliest use of single-layer auto-encoders for building deep architectures by stacking them (Bengio *et al.*, 2007) considered the idea of *tying* the encoder weights and decoder weights to restrict capacity as well as the idea of introducing a form of *sparsity regularization* (Ranzato *et al.*, 2007). Several ways of introducing sparsity in the representation learned by auto-encoders have then been proposed, some by penalizing the hidden unit biases (making these additive offset parameters more negative) (Ranzato *et al.*, 2007; Lee *et al.*, 2008; Goodfellow *et al.*, 2009; Larochelle and Bengio, 2008) and some by directly penalizing the output of the hidden unit activations (making them closer to their saturating value at 0) (Ranzato *et al.*, 2008; Le *et al.*, 2011a; Zou *et al.*, 2011). Note that penalizing the bias runs the danger that the weights could compensate for the bias, which could hurt the numerical optimization of parameters. When directly penalizing the hidden unit outputs, several variants can be found in the literature, but no clear comparative analysis has been published to evaluate which one works better. Although the L1 penalty (i.e., simply the sum of output elements h_j in the case of sigmoid non-linearity) would seem the most natural (because of its use in sparse coding), it is used in few papers involving sparse auto-encoders. A close cousin of the L1 penalty is the Student-t penalty ($\log(1+h_j^2)$), originally proposed for sparse coding (Olshausen and Field, 1997). Several papers penalize the *average* output $\bar{h}_j$ (e.g. over a minibatch), and instead of pushing it to 0, encourage it to approach a fixed target, either through a mean-square error penalty, or maybe more sensibly (because h_j behaves like a probability), a Kullback-Liebler divergence with respect to the binomial distribution with probability ρ: $-\rho \log \bar{h}_j - (1-\rho)\log(1-\bar{h}_j)$+constant, e.g., with $\rho = 0.05$.

7.2.2 *Denoising Auto-Encoders*

Vincent *et al.* (2008, 2010) proposed altering the training objective in Equation 19 from mere reconstruction to that of *denoising* an artificially corrupted input, i.e. learning to reconstruct the clean input from a corrupted version. Learning the identity is no longer enough: the learner must capture the structure of the input distribution in order to optimally undo the effect of the corruption process, with the reconstruction essentially being a nearby but higher density point than the corrupted input. Figure 3 illustrates that the denoising auto-encoder is learning a reconstruction function that corresponds to a vector field pointing towards high-density regions (the manifold where examples concentrate).

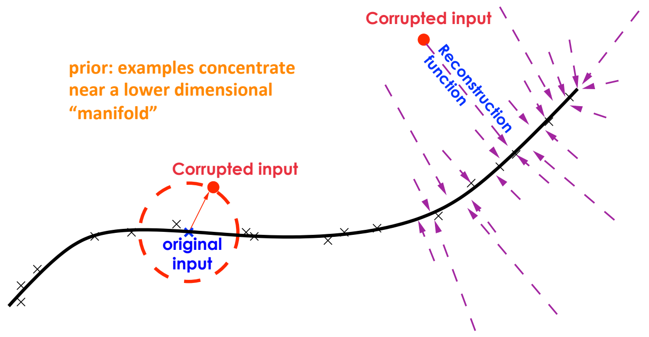

Fig. 3. When the data concentrate near a lower-dimensional manifold, the corruption vector is most of the time almost orthogonal to the manifold, and the reconstruction function learns to denoise, map from low-probability configurations (corrupted inputs) to high-probability ones (original inputs), creating a kind of vector field aligned with the score (derivative of the estimated density).

Formally, the objective optimized by such a Denoising

Auto-Encoder (DAE) is:

$$\mathcal{J}_{\text{DAE}} = \sum_t \mathbb{E}_{q(\tilde{x}|x^{(t)})}\left[L(x^{(t)}, g_\theta(f_\theta(\tilde{x})))\right] \quad (22)$$

where $\mathbb{E}_{q(\tilde{x}|x^{(t)})}[\cdot]$ denotes the expectation over corrupted examples $\tilde{x}$ drawn from corruption process $q(\tilde{x}|x^{(t)})$. In practice this is optimized by stochastic gradient descent, where the stochastic gradient is estimated by drawing one or a few corrupted versions of $x^{(t)}$ each time $x^{(t)}$ is considered. Corruptions considered in Vincent *et al.* (2010) include additive isotropic Gaussian noise, salt and pepper noise for gray-scale images, and masking noise (salt or pepper only). Qualitatively better features are reported, resulting in improved classification performance, compared to basic auto-encoders, and similar or better than that obtained with RBMs. Chen *et al.* (2012) show that a simpler alternative with a closed form solution can be obtained when restricting to a *linear* auto-encoder and have successfully applied it to domain adaptation.

The analysis in Vincent (2011) relates the denoising auto-encoder criterion to energy-based probabilistic models: denoising auto-encoders basically learn in $r(\tilde{x}) - \tilde{x}$ a vector pointing in the direction of the estimated *score* i.e., $\frac{\partial \log p(\tilde{x})}{\partial \tilde{x}}$, as illustrated in Figure 3. In the special case of linear reconstruction and squared error, Vincent (2011) shows that DAE training amounts to learning an energy-based model, whose energy function is very close to that of a GRBM, using a regularized variant of the *score matching* parameter estimation technique (Hyvärinen, 2005a; Hyvärinen, 2008; Kingma and LeCun, 2010) termed *denoising score matching* (Vincent, 2011). Previously, Swersky (2010) had shown that training GRBMs with *score matching* was equivalent to training a regular (non-denoising) auto-encoder with an additional regularization term, while, following up on the theoretical results in Vincent (2011), Swersky *et al.* (2011) showed the practical advantage of the denoising criterion to implement score matching efficiently.

7.2.3 *Contractive Auto-Encoders*

Contractive Auto-Encoders (CAE) proposed by Rifai *et al.* (2011a) follow up on Denoising Auto-Encoders (DAE) and share a similar motivation of learning robust representations. CAEs achieve this by adding an analytic *contractive penalty* term to the basic auto-encoder of Equation 19. This term is the Frobenius norm of the encoder's Jacobian, and results in penalizing the *sensitivity* of learned features to infinitesimal changes of the input.

Let $J(x) = \frac{\partial f_\theta}{\partial x}(x)$ the Jacobian matrix of the encoder evaluated at x. The CAE's training objective is the following:

$$\mathcal{J}_{\text{CAE}} = \sum_t L(x^{(t)}, g_\theta(f_\theta(x^{(t)}))) + \lambda \left\| J(x^{(t)}) \right\|_F^2 \quad (23)$$

where λ is a hyper-parameter controlling the strength of the regularization.

For an affine sigmoid encoder, the contractive penalty term is easy to compute:

$$\begin{aligned} J_j(x) &= f_\theta(x)_j(1 - f_\theta(x)_j)W_j \\ \left\| J(x^{(t)}) \right\|^2 &= \sum_j (f_\theta(x)_j(1 - f_\theta(x)_j))^2 \|W_j\|^2 \end{aligned} \quad (24)$$

There are at least three notable differences with DAEs, which may be partly responsible for the better performance that CAE features seem to empirically demonstrate: a) the sensitivity of the *features* is penalized[14] directly rather than the sensitivity of the *reconstruction*; b) penalty is analytic rather than stochastic: an efficiently computable expression replaces what might otherwise require d_x corrupted samples to size up (i.e. the sensitivity in d_x directions); c) a hyper-parameter λ allows a fine control of the trade-off between reconstruction and robustness (while the two are mingled in a DAE). Note however that there is a tight connection between the DAE and the CAE: as shown in (Bengio *et al.*, 2012b) a DAE with small corruption noise can be seen (through a Taylor expansion) as a type of contractive auto-encoder where the contractive penalty is on the whole reconstruction function rather than just on the encoder[15].

A potential disadvantage of the CAE's analytic penalty is that it amounts to only encouraging robustness to *infinitesimal* changes of the input. This is remedied by a further extension proposed in Rifai *et al.* (2011b) and termed CAE+H, that penalizes all higher order derivatives, in an efficient stochastic manner, by adding a third term that encourages $J(x)$ and $J(x + \epsilon)$ to be close:

$$\begin{aligned} \mathcal{J}_{\text{CAE+H}} = &\sum_t L(x^{(t)}, g_\theta(x^{(t)})) + \lambda \left\| J(x^{(t)}) \right\|_F^2 \\ &+ \gamma \mathbb{E}_\epsilon \left[\|J(x) - J(x + \epsilon)\|_F^2 \right] \end{aligned} \quad (25)$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, and γ is the associated regularization strength hyper-parameter. As for the DAE, the training criterion is optimized by stochastic gradient descent, whereby the expectation is approximated by drawing several corrupted versions of $x^{(t)}$.

Note that the DAE and CAE have been successfully used to win the final phase of the Unsupervised and Transfer Learning Challenge (Mesnil *et al.*, 2011). Note also that the representation learned by the CAE tends to be *saturated* rather than sparse, i.e., most of the hidden units are near the extremes of their range (e.g. 0 or 1), and their derivative $\frac{\partial h_i(x)}{\partial x}$ is tiny. The non-saturated units are few and sensitive to the inputs, with their associated filters (hidden unit weight vector) together forming a basis explaining the local changes around x, as discussed in Section 8.2. Another way to get saturated (i.e. nearly binary) units (for the purpose of hashing) is *semantic hashing* (Salakhutdinov and Hinton, 2007).

7.2.4 *Predictive Sparse Decomposition*

Sparse coding (Olshausen and Field, 1997) may be viewed as a kind of auto-encoder that uses a linear decoder with a squared reconstruction error, but whose non-parametric *encoder* f_θ performs the comparatively non-trivial and relatively costly minimization of Equation. 2, which entails an iterative optimization.

A practically successful variant of sparse coding and auto-encoders, named *Predictive Sparse Decomposition* or

14. i.e., the robustness of the representation is encouraged.

15. but note that in the CAE, the decoder weights are tied to the encoder weights, to avoid degenerate solutions, and this should also make the decoder contractive.

PSD (Kavukcuoglu *et al.*, 2008) replaces that costly and highly non-linear encoding step by a fast non-iterative approximation during recognition (computing the learned features). PSD has been applied to object recognition in images and video (Kavukcuoglu *et al.*, 2009, 2010; Jarrett *et al.*, 2009; Farabet *et al.*, 2011), but also to audio (Henaff *et al.*, 2011), mostly within the framework of multi-stage convolutional and hierarchical architectures (see Section 11.2). The main idea can be summarized by the following equation for the training criterion, which is simultaneously optimized with respect to the hidden codes (representation) $h^{(t)}$ and with respect to the parameters (W, α):

$$\mathcal{J}_{\text{PSD}} = \sum_t \lambda \|h^{(t)}\|_1 + \|x^{(t)} - W h^{(t)}\|_2^2 + \|h^{(t)} - f_\alpha(x^{(t)})\|_2^2 \quad (26)$$

where $x^{(t)}$ is the input vector for example t, $h^{(t)}$ is the optimized hidden code for that example, and $f_\alpha(\cdot)$ is the encoding function, the simplest variant being

$$f_\alpha(x^{(t)}) = \tanh(b + W^T x^{(t)}) \quad (27)$$

where the encoding weights are the transpose of the decoding weights, but many other variants have been proposed, including the use of a shrinkage operation instead of the hyperbolic tangent (Kavukcuoglu *et al.*, 2010). Note how the L1 penalty on h tends to make them sparse, and notice that it is the same criterion as sparse coding with dictionary learning (Eq. 3) except for the additional constraint that one should be able to approximate the sparse codes h with a parametrized encoder $f_\alpha(x)$. One can thus view PSD as an approximation to sparse coding, where we obtain a fast approximate encoding process as a side effect of training. In practice, once PSD is trained, object representations used to feed a classifier are computed from $f_\alpha(x)$, which is very fast, and can then be further optimized (since the encoder can be viewed as one stage or one layer of a trainable multi-stage system such as a feedforward neural network).

PSD can also be seen as a kind of auto-encoder (there is an encoder $f_\alpha(\cdot)$ and a decoder W) where, instead of being tied to the output of the encoder, the codes h are given some freedom that can help to further improve reconstruction. One can also view the encoding penalty added on top of sparse coding as a kind of regularizer that forces the sparse codes to be nearly computable by a smooth and efficient encoder. This is in contrast with the codes obtained by complete optimization of the sparse coding criterion, which are highly non-smooth or even non-differentiable, a problem that motivated other approaches to smooth the inferred codes of sparse coding (Bagnell and Bradley, 2009), so a sparse coding stage could be jointly optimized along with following stages of a deep architecture.

8 Representation Learning as Manifold Learning

Another important perspective on representation learning is based on the geometric notion of manifold. Its premise is the *manifold hypothesis* (Cayton, 2005; Narayanan and Mitter, 2010), according to which real-world data presented in high dimensional spaces are expected to concentrate in the vicinity of a manifold $\mathcal{M}$ of much lower dimensionality $d_\mathcal{M}$, embedded in high dimensional input space $\mathbb{R}^{d_x}$. This can be a potentially powerful prior for representation learning for AI tasks. As soon as there is a notion of ”representation” then one can think of a manifold by considering the *variations* in input space, which are captured by or reflected (by corresponding changes) in the learned representation. To first approximation, some directions are well preserved (they are the tangent directions of the manifold) while others aren’t (they are directions orthogonal to the manifolds). With this perspective, the primary unsupervised learning task is then seen as modeling the structure of the data-supporting manifold[16]. The associated *representation* being learned corresponds to an intrinsic coordinate system on the embedded manifold. The archetypal manifold modeling algorithm is, not surprisingly, also the archetypal low dimensional representation learning algorithm: Principal Component Analysis. PCA models a *linear* manifold. It was initially devised by Pearson (1901) precisely with the objective of finding the closest linear manifold (specifically a line or a plane) to a cloud of data points. The principal components, i.e. the *representation* $f_\theta(x)$ that PCA yields for an input point x, uniquely locates its projection on that manifold: it corresponds to intrinsic coordinates on the manifold. Data manifold for complex real world domains are however expected to be *strongly non-linear*. Their modeling is sometimes approached as patchworks of locally linear tangent spaces (Vincent and Bengio, 2003; Brand, 2003). The large majority of algorithms built on this geometric perspective adopt a non-parametric approach, based on a training set nearest neighbor graph (Schölkopf *et al.*, 1998; Roweis and Saul, 2000; Tenenbaum *et al.*, 2000; Brand, 2003; Belkin and Niyogi, 2003; Donoho and Grimes, 2003; Weinberger and Saul, 2004; Hinton and Roweis, 2003; van der Maaten and Hinton, 2008). In these non-parametric approaches, each high-dimensional training point has its own set of free low-dimensional *embedding* coordinates, which are optimized so that certain properties of the neighborhood graph computed in original high dimensional input space are best preserved. These methods however do not directly learn a parametrized feature extraction function $f_\theta(x)$ applicable to new test points[17], which seriously limits their use as feature extractors, except in a transductive setting. Comparatively few non-linear manifold learning methods have been proposed, that learn a *parametric map* that can directly compute a representation for new points; we will focus on these.

8.1 Learning a parametric mapping based on a neighborhood graph

The non-parametric manifold learning algorithms we just mentioned are all based on a training set neighborhood graph, typically derived from pairwise Euclidean distances between training points. Some of them are not too difficult to modify from non-parametric to instead learn a *parametric mapping* f_θ,

16. What is meant by data manifold is actually a loosely defined notion: data points need not strictly lie on it, but the probability density is expected to fall off sharply as one moves away from the “manifold” (which may actually be constituted of several possibly disconnected manifolds with different intrinsic dimensionality).

17. For several of these techniques, representations for new points can be computed using the Nyström approximation as has been proposed as an extension in (Bengio *et al.*, 2004), but this remains cumbersome and computationally expensive.

that will be applicable to new points. The principle is simple: instead of having *free* low-dimensional embedding coordinate "parameters" for each training point, these coordinates are now obtained through an explicitly parametrized function on input-space coordinates, whose parameters are to be learned. The same optimization objective as in the non-parametric version can be minimized, through gradient descent: now instead of gradient descent updates on the embedding coordinates, gradients are backpropagated further to the parameters of that mapping function. Thus a parametric version of the very successful non-parametric manifold embedding algorithm t-SNE (van der Maaten and Hinton, 2008) has been proposed in (van der Maaten, 2009), and could be directly applied to learning a direct parametric encoding.

Another interesting approach, that learns a direct encoding while taking into account the manifold hypothesis through a neighborhood graph is Semi-Supervised Embedding (Weston *et al.*, 2008). Here a deep parametrized neural network architecture simultaneously learns a manifold embedding and a classifier. While optimizing the supervised classification cost, the training criterion also uses training set neighbors of each training example to encourage intermediate layers of representation to be *invariant* when changing the training example for a neighbor.

The more reduced and tightly controlled number of free parameters in such methods, compared to their pure non-parametric counterparts, forces the models to generalize the manifold shape non-locally (Bengio *et al.*, 2006b), which, provided that generalization is valid, can translate into better features and final performance (van der Maaten and Hinton, 2008).

Yet basing the modeling of manifolds on training set neighborhood relationships might be risky statistically in high dimensional spaces (sparsely populated due to the curse of dimensionality) as e.g. most Euclidean nearest neighbors risk having too little in common semantically. The neareset neighbor graph is simply not enough densely populated to map out satisfyingly the wrinkles of the target manifold. It can also become problematic computationally to consider all pairs of data points[18], which scales quadratically with training set size.

8.2 Learning a non-linear manifold through a coding scheme

We now turn to manifold interpretations of learning techniques that are not based on training set neighbor searches. Let us begin with PCA, seen as an encoding scheme. In PCA, the same basis vectors are used to project any input point x. The sensitivity of the extracted components (the code) to input changes in the direction of these vectors is the same regardless of position x. The tangent space is the same everywhere along the linear manifold. By contrast, for a non-linear manifold, the tangent space is expected to change directions as we move, as illustrated by the tangent plane in Figure 6. In non-linear representation-learning algorithms it is convenient to think about the *local variations* in the representation as the input x is varied *on the manifold*, i.e., as we move from a high-probability example to a very close one in input space. As we will discuss below, the first derivative of the mapping from input to representation (the encoder) therefore specifies the shape of the manifold (its tangent plane) around an example x lying on it. If the density was really concentrated on the manifold, and the encoder had captured that, we would find the derivatives to be non-zero only in the directions spanned by the tangent plane.

18. Even if pairs are picked stochastically, many must be considered before obtaining one that weighs significantly on the optimization objective.

Let us consider sparse-coding in this light: parameter matrix W may be interpreted as a dictionary of input directions from which a *different subset* will be picked to model the local tangent space at an x on the manifold. That subset corresponds to the active, i.e. non-zero, features for input x. Note that non-zero component h_i will be sensitive to small changes of the input in the direction of the associated weight vector $W_{:,i}$, whereas inactive features are more likely to be stuck at 0 until a significant displacement has taken place in input space.

The *Local Coordinate Coding* (LCC) algorithm (Yu *et al.*, 2009) is very similar to sparse coding, but is explicitly derived from a manifold perspective. Using the same notation as that of sparse-coding in Equation 2, LCC replaces regularization term $\|h^{(t)}\|_1 = \sum_j |h_j^{(t)}|$ yielding objective

$$\mathcal{J}_{\text{LCC}} = \sum_t \left(\|x^{(t)} - Wh^{(t)}\|_2^2 + \lambda \sum_j |h_j^{(t)}| \|W_{:,j} - x^{(t)}\|^{1+p} \right) \tag{28}$$

This is identical to sparse-coding when $p = -1$, but with larger p it encourages the active *anchor points* for $x^{(t)}$ (i.e. the codebook vectors $W_{:,j}$ with non-negligible $|h_j^{(t)}|$ that are combined to reconstruct $x^{(t)}$) to be not too far from $x^{(t)}$, hence the *local* aspect of the algorithm. An important theoretical contribution of Yu *et al.* (2009) is to show that that any Lipschitz-smooth function $\phi : \mathcal{M} \to \mathbb{R}$ defined on a smooth nonlinear manifold $\mathcal{M}$ embedded in $\mathbb{R}^{d_x}$, can be well approximated by a globally *linear function* with respect to the resulting coding scheme (i.e. linear in h), where the accuracy of the approximation and required number d_h of anchor points depend on $d_{\mathcal{M}}$ rather than d_x. This result has been further extended with the use of local tangent directions (Yu and Zhang, 2010), as well as to multiple layers (Lin *et al.*, 2010).

Let us now consider the efficient non-iterative "feed-forward" encoders f_θ, used by PSD and the auto-encoders reviewed in Section 7.2, that are in the form of Equation 20 or 27.The computed representation for x will be only significantly sensitive to input space directions associated with non-saturated hidden units (see e.g. Eq. 24 for the Jacobian of a sigmoid layer). These directions to which the representation is significantly sensitive, like in the case of PCA or sparse coding, may be viewed as spanning the tangent space of the manifold at training point x.

Rifai *et al.* (2011a) empirically analyze in this light the singular value spectrum of the Jacobian (derivatives of representation vector with respect to input vector) of a trained CAE. Here the SVD provides an ordered orthonormal basis of most sensitive directions. The spectrum is sharply decreasing, indicating a relatively small number of significantly sensitive directions. This is taken as empirical evidence that the

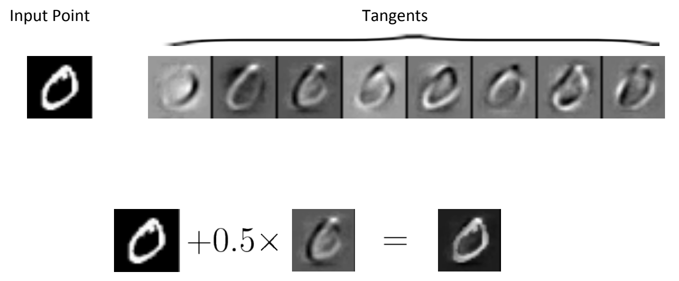

Fig. 4. The tangent vectors to the high-density manifold as estimated by a Contractive Auto-Encoder (Rifai *et al.*, 2011a). The original input is shown on the top left. Each tangent vector (images on right side of first row) corresponds to a plausible additive deformation of the original input, as illustrated on the second row, where a bit of the 3rd singular vector is added to the original, to form a translated and deformed image. Unlike in PCA, the tangent vectors are different for different inputs, because the estimated manifold is highly non-linear.

CAE indeed modeled the tangent space of a low-dimensional manifold. The leading singular vectors form a basis for the tangent plane of the estimated manifold, as illustrated in Figure 4. The CAE criterion is believed to achieve this thanks to its two opposing terms: the isotropic contractive penalty, that encourages the representation to be equally insensitive to changes in any input directions, and the reconstruction term, that pushes different training points (in particular neighbors) to have a different representation (so they may be reconstructed accurately), thus counteracting the isotropic contractive pressure only in directions tangent to the manifold.

Note that analyzing learned representations through the lens of the spectrum of the Jacobian and relating it to the notion of tangent space of a manifold is feasible, whenever the mapping is differentiable, and regardless of how it was learned, whether as direct encoding (as in auto-encoder variants), or derived from latent variable inference (as in sparse coding or RBMs). Exact low dimensional manifold models (like PCA) would yield non-zero singular values associated to directions along the manifold, and exact zeros for directions orthogonal to the manifold. But in smooth models like the contractive auto-encoder or the RBM we will instead have large versus relatively small singular values (as opposed to non-zero versus exactly zero).

8.3 Leveraging the modeled tangent spaces

The local tangent space, at a point along the manifold, can be thought of capturing *locally* valid transformations that were prominent in the training data. For example Rifai *et al.* (2011c) examine the tangent directions extracted with an SVD of the Jacobian of CAEs trained on digits, images, or text-document data: they appear to correspond to small translations or rotations for images or digits, and to substitutions of words within a same theme for documents. Such very local transformations along a data manifold are not expected to change class identity. To build their Manifold Tangent Classifier (MTC), Rifai *et al.* (2011c) then apply techniques such as *tangent distance* (Simard *et al.*, 1993) and *tangent propagation* (Simard *et al.*, 1992), that were initially developed to build classifiers that are insensitive to input deformations provided as prior domain knowledge. Now these techniques are applied using the local leading tangent directions extracted by a CAE, i.e. not using *any* prior domain knowledge (except the broad prior about the existence of a manifold). This approach set a new record for MNIST digit classification among prior-knowledge free approaches[19].

9 Connections between Probabilistic and Direct Encoding models

The standard likelihood framework for probabilistic models decomposes the training criterion for models with parameters θ in two parts: the log-likelihood $\log P(x|\theta)$ (or $\log P(x|h,\theta)$ with latent variables h), and the prior $\log P(\theta)$ (or $\log P(h|\theta) + \log P(\theta)$ with latent variables).

9.1 PSD: a probabilistic interpretation

In the case of the PSD algorithm, a connection can be made between the above standard probabilistic view and the direct encoding computation graph. In this view, the probabilistic model of PSD is the same directed generative model $P(x|h)$ of sparse coding (Section 6.1.3), which only accounts for the decoder. The encoder is viewed as an approximate inference mechanism used to guess $P(h|x)$ and initialize a MAP iterative inference (where the sparse prior $P(h)$ is taken into account). However, note that in PSD, the encoder is *trained jointly with the decoder*, rather than simply taking the end result of iterative inference as a target to approximate. An interesting view[20] to integrate this fact is that the encoder is a parametric approximation for the MAP solution of a variational lower bound on the joint log-likelihood. When MAP learning is viewed as a special case of variational learning (where the approximation of the joint log-likelihood is with a dirac distribution located at the MAP solution), the variational recipe tells us to simultaneously improve the likelihood (reduce reconstruction error) and improve the variational approximation (reduce the discrepancy between the encoder output and the latent variable value). Hence PSD is an interesting case of representation learning algorithm that sits at the intersection of probabilistic models (with latent variables) and direct encoding methods (which directly parametrize the mapping from input to representation). RBMs also sit at the intersection because their particular parametrization includes an explicit mapping from input to representation, thanks to the restricted connectivity between hidden units. However, this nice property does not extend to their natural deep generalizations, i.e., Deep Boltzmann Machines, discussed in Section 10.2.

9.2 Regularized Auto-Encoders Capture Local Statistics of the Density

Can we also say something about the probabilistic interpretation of regularized auto-encoders, including sparse

19. It yielded 0.81% error rate using the full MNIST training set, with no prior deformations, and no convolution.

20. suggested by Ian Goodfellow, personal communication

auto-encoders, denoising auto-encoders, and contractive auto-encoders? Their training criterion does not fit the standard likelihood framework because this would require a kind of prior (e.g. we want a sparse or contractive or robust representation) that is *data-dependent*.

An interesting hypothesis emerges to answer that question, out of recent theoretical results (Vincent, 2011; Bengio *et al.*, 2012b): their training criterion, instead of being a form of maximum likelihood, corresponds to a different inductive principle, such as *score matching*. The score matching connection is discussed in Section 7.2.2 and has been shown for a particular parametrization of Denoising Auto-Encoder and equivalent Gaussian RBM (Vincent, 2011). The work in Bengio *et al.* (2012b) generalizes this idea to a broader class of parametrizations (arbitrary encoders and decoders), and shows that by regularizing the auto-encoder so that it be contractive (which is the case not only of contractive auto-encoders but also of denoising and sparse ones), one obtains that the reconstruction function and its derivative *estimate local statistics* of the underlying data-generative distribution, such as the local mean (the mean in a small ball around each point), the local covariance, and the first and second derivatives of the estimated density. This view can actually be exploited to successfully *sample* from auto-encoders, as shown in Rifai *et al.* (2012); Bengio *et al.* (2012b). The proposed sampling algorithms are MCMCs similar to Langevin MCMC, using not just the estimated first derivative of the density but also the estimated second derivative, so as to stay close to manifolds near which the density concentrates.

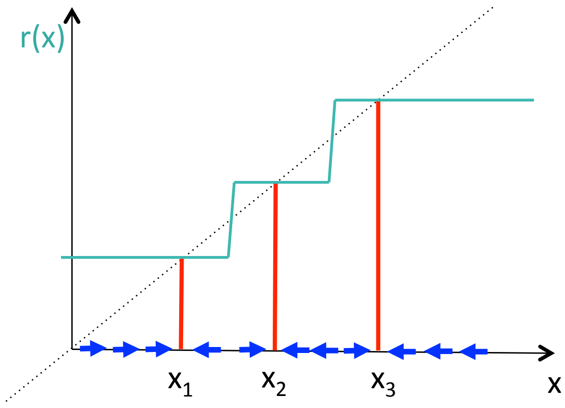

Fig. 5. The reconstruction function $r(x)$ (in green) learned by an autoencoder on a 1-dimensional input with high capacity, minimizing reconstruction error *at the training examples* x_i (with in $r(x_i)$ in red) while trying to be as constant as possible otherwise. The dotted line is the identity reconstruction (which might be obtained without the regularizer). The blue arrows shows the vector field of $r(x) - x$ pointing towards high density peaks as estimated by the model, and estimating the score (log-density derivative).

This interpretation connects well with the geometric perspective introduced in Section 8. The regularization effects (e.g., due to a sparsity regularizer, a contractive regularizer, or the denoising criterion) asks the learned representation to be as insensitive as possible to the input, while minimizing reconstruction error on the training examples forces the representation to contain just enough information to distinguish them. The solution is that variations along the high-density manifolds are preserved while other variations are compressed. It means that the reconstruction function should be as constant as possible while reproducing training examples, i.e., that points near a training example should be mapped to that training example, as illustrated in Figure 5. This also means that the reconstruction function should map an input towards the nearest point manifold, i.e., the difference between reconstruction and input is a vector aligned with the estimated score (the derivative of the log-density with respect to the input). When the score is zero (on the manifold), we have to look towards the second derivative of the log-density or of the energy (and the first derivative of the reconstruction function). The directions of smallest second derivatives of the log-density are those where the density remains high (where the first derivative remains close to 0) and correspond to moving on the manifold.

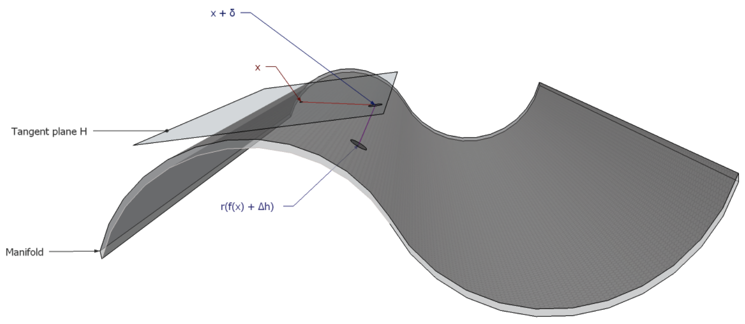

Fig. 6. Illustration of the sampling procedure for regularized auto-encoders (Rifai *et al.*, 2012; Bengio *et al.*, 2012b): Each MCMC step consists in adding noise δ mostly in the directions of the estimated manifold tangent plane and projecting back towards the manifold (high-density regions) by performing a reconstruction step.

As illustrated in Figure 6, the basic idea of the auto-encoder sampling algorithms in Rifai *et al.* (2012); Bengio *et al.* (2012b) is to make MCMC moves where one (a) moves toward the manifold by following the density gradient (i.e., applying a reconstruction) and (b) adds noise in the directions of the leading singular vectors of the reconstruction (or encoder) Jacobian, corresponding to those associated with smallest second derivative of the log-density.

9.3 Learning Approximate Inference

Let us now consider from closer how a representation is computed in probabilistic models with latent variables, when iterative inference is required. There is a computation graph (possibly with random number generation in some of the nodes, in the case of MCMC) that maps inputs to representation, and in the case of deterministic inference (e.g., MAP inference or variational inference), that function could be optimized directly. This is a way to generalize PSD that has been explored in recent work on probabilistic models at the intersection of inference and learning (Bagnell and Bradley, 2009; Gregor and LeCun, 2010b; Grubb and Bagnell, 2010; Salakhutdinov and Larochelle, 2010; Stoyanov *et al.*, 2011;

Eisner, 2012), where a central idea is that instead of using a *generic* inference mechanism, one can use one that is *learned* and is more efficient, taking advantage of the specifics of the type of data on which it is applied.

9.4 Sampling Challenges

A troubling challenge with many probabilistic models with latent variables like most Boltzmann machine variants is that good MCMC sampling is required as part of the learning procedure, but that *sampling becomes extremely inefficient (or unreliable) as training progresses* because the *modes of the learned distribution become sharper*, making *mixing between modes very slow*. Whereas initially during training a learner assigns mass almost uniformly, as training progresses, its entropy decreases, approaching the entropy of the target distribution as more examples and more computation are provided. According to our Manifold and Natural Clustering priors of Section 3.1, the target distribution has sharp modes (manifolds) separated by extremely low density areas. Mixing then becomes more difficult because MCMC methods, by their very nature, tend to make small steps to nearby high-probability configurations. This is illustrated in Figure 7.

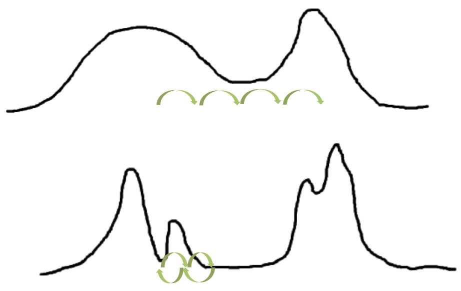
Fig. 7. Top: early during training, MCMC mixes easily because the estimated distribution has high entropy and puts enough mass everywhere for small-steps movements (MCMC) to go from mode to mode. Bottom: later on, training relying on good mixing can stall because estimated modes are separated by long low-density deserts.

Bengio *et al.* (2012a) suggest that deep representations could help mixing between such well separated modes, based on both theoretical arguments and on empirical evidence. The idea is that if higher-level representations disentangle better the underlying abstract factors, then small steps in this abstract space (e.g., swapping from one category to another) can easily be done by MCMC. The high-level representations can then be mapped back to the input space in order to obtain input-level samples, as in the Deep Belief Networks (DBN) sampling algorithm (Hinton *et al.*, 2006a). This has been demonstrated both with DBNs and with the newly proposed algorithm for sampling from contracting and denoising auto-encoders (Rifai *et al.*, 2012; Bengio *et al.*, 2012b). This observation alone does not suffice to solve the problem of training a DBN or a DBM, but it may provide a crucial ingredient, and it makes it possible to consider successfully sampling from deep models trained by procedures that do not require an MCMC, like the stacked regularized auto-encoders used in Rifai *et al.* (2012).

9.5 Evaluating and Monitoring Performance

It is always possible to evaluate a feature learning algorithm in terms of its usefulness with respect to a particular task (e.g. object classification), with a predictor that is fed or initialized with the learned features. In practice, we do this by saving the features learned (e.g. at regular intervals during training, to perform early stopping) and training a cheap classifier on top (such as a linear classifier). However, training the final classifier can be a substantial computational overhead (e.g., supervised fine-tuning a deep neural network takes usually more training iterations than the feature learning itself), so we may want to avoid having to train a classifier for every training iteration of the unsupervised learner and every hyper-parameter setting. More importantly this may give an incomplete evaluation of the features (what would happen for other tasks?). All these issues motivate the use of methods to monitor and evaluate purely unsupervised performance. This is rather easy with all the auto-encoder variants (with some caution outlined below) and rather difficult with the undirected graphical models such as the RBM and Boltzmann machines.

For auto-encoder and sparse coding variants, test set reconstruction error can readily be computed, but by itself may be misleading because *larger capacity* (e.g., more features, more training time) tends to systematically lead to lower reconstruction error, even on the test set. Hence it cannot be used reliably for selecting most hyper-parameters. On the other hand, *denoising* reconstruction error is clearly immune to this problem, so that solves the problem for denoising auto-encoders. It is not clear and remains to be demonstrated empirically or theoretically if this may also be true of the training criteria for sparse auto-encoders and contractive auto-encoders.

For RBMs and some (not too deep) Boltzmann machines, one option is the use of Annealed Importance Sampling (Murray and Salakhutdinov, 2009) in order to estimate the partition function (and thus the test log-likelihood). Note that this estimator can have high variance and that it becomes less reliable (variance becomes too large) as the model becomes more interesting, with larger weights, more non-linearity, sharper modes and a sharper probability density function (see our previous discussion in Section 9.4). Another interesting and recently proposed option for RBMs is to *track* the partition function during training (Desjardins *et al.*, 2011), which could be useful for early stopping and reducing the cost of ordinary AIS. For toy RBMs (e.g., 25 hidden units or less, or 25 inputs or less), the exact log-likelihood can also be computed analytically, and this can be a good way to debug and verify some properties of interest.

10 Global Training of Deep Models

One of the most interesting challenges raised by deep architectures is: *how should we jointly train all the levels*? In the previous section we have only discussed how single-layer models could be combined to form a deep model with a joint training criterion. Here we consider joint training of all the levels and the difficulties that may arise. This follows up on Section 4, where we focused on how to combine single-layer learners into deep architectures.

10.1 On the Challenge of Training Deep Architectures

Higher-level abstraction means more non-linearity. It means that two nearby configurations of the input may have to be interpreted very differently because a few surface details change the underlying semantics, whereas most other changes in the surface details would not change the underlying semantics. The representations associated with input manifolds may be complex because these functions may have to unfold and distort input manifolds that generally have complicated shapes into spaces where distributions are much simpler, where relations between factors are simpler, maybe even linear or involving many (conditional) independencies. Our expectation is that modeling the joint distribution between high-level abstractions and concepts should be much easier in the sense of requiring much less data to learn. The hard part is learning the representation. The price to pay may be that these more abstract representation functions (mapping input to representation) involve a more challenging training task, because of the added non-linearity.

It is only since 2006 that researchers have seriously investigated ways to train deep architectures, to the exception of the convolutional networks (LeCun *et al.*, 1998b). The first realization was that unsupervised or supervised layer-wise training was easier, and that this could be taken advantage of by stacking single-layer models into deeper ones, as discussed at length in Section 4.

It is interesting to ask *why does the layerwise unsupervised pre-training procedure sometimes help a supervised learner* (Erhan *et al.*, 2010b). There seems to be a more general principle at play [21] of *guiding the training of intermediate representations*, which may be easier than trying to learn it all in one go. This is nicely related with the curriculum learning idea (Bengio *et al.*, 2009) that it may be much easier to learn simpler concepts first and then build higher-level ones on top of simpler ones. This is also coherent with the success of several deep learning algorithms that provide some such guidance for intermediate representations, like Semi-Supervised Embedding (Weston *et al.*, 2008).

The question of why unsupervised pre-training could be helpful was extensively studied (Erhan *et al.*, 2010b), trying to dissect the answer into a *regularization effect* and an *optimization effect*. The regularization effect is clear from the experiments where the stacked RBMs or denoising auto-encoders are used to initialize a supervised classification neural network (Erhan *et al.*, 2010b). It may simply come from the use of unsupervised learning to bias the learning dynamics and initialize it in the *basin of attraction* of a "good" local minimum (of the training criterion), where "good" is in terms of generalization error. The underlying hypothesis exploited by this procedure is that some of the features or latent factors that are good at capturing the leading variations in the input distribution are also good at capturing the variations in the target output random variables of interest (e.g., classes). The optimization effect is more difficult to tease out because the top two layers of a deep neural net can just overfit the training set whether the lower layers compute useful features or not, but there are several indications that optimizing the lower levels with respect to a supervised training criterion is challenging.

21. First communicated to us by Leon Bottou

One such indication is that changing the numerical conditions of the optimization procedure can have a profound impact on the joint training of a deep architecture, for example changing the initialization range and changing the type of non-linearity used (Glorot and Bengio, 2010), much more so than with shallow architectures. One hypothesis to explain some of the difficulty in the optimization of deep architectures is centered on the singular values of the Jacobian matrix associated with the transformation from the features at one level into the features at the next level (Glorot and Bengio, 2010). If these singular values are all small (less than 1), then the mapping is contractive in every direction and *gradients would vanish* when propagated backwards through many layers. This is a problem already discussed for *recurrent neural networks* (Bengio *et al.*, 1994), which can be seen as very deep networks with shared parameters at each layer, when unfolded in time. This optimization difficulty has motivated the exploration of second-order methods for deep architectures and recurrent networks, in particular Hessian-free second-order methods (Martens, 2010; Martens and Sutskever, 2011). Unsupervised pre-training has also been proposed to help training recurrent networks and temporal RBMs (Sutskever *et al.*, 2009), i.e., at each time step there is a local signal to guide the discovery of good features to capture in the state variables: model with the current state (as hidden units) the joint distribution of the previous state and the current input. Natural gradient (Amari, 1998) methods that can be applied to networks with millions of parameters (i.e. with good scaling properties) have also been proposed (Le Roux *et al.*, 2008b). Cho *et al.* (2011) proposes to use adaptive learning rates for RBM training, along with a novel and interesting idea for a gradient estimator that takes into account the invariance of the model to flipping hidden unit bits and inverting signs of corresponding weight vectors. At least one study indicates that the choice of initialization (to make the Jacobian of each layer closer to 1 across all its singular values) could substantially reduce the training difficulty of deep networks (Glorot and Bengio, 2010) and this is coherent with the success of the initialization procedure of Echo State Networks (Jaeger, 2007), as recently studied by Sutskever (2012). There are also several experimental results (Glorot and Bengio, 2010; Glorot *et al.*, 2011a; Nair and Hinton, 2010) showing that the choice of hidden units non-linearity could influence both training and generalization performance, with particularly interesting results obtained with sparse rectifying units (Jarrett *et al.*, 2009; Nair and Hinton, 2010; Glorot *et al.*, 2011a; Krizhevsky *et al.*, 2012). Another promising idea to improve the conditioning of neural network training is to nullify the average value and slope of each hidden unit output (Raiko *et al.*, 2012), and possibly locally normalize magnitude as well (Jarrett *et al.*, 2009). The debate still rages between using online methods such as stochastic gradient descent and using second-order methods on large minibatches (of several thousand examples) (Martens, 2010; Le *et al.*, 2011a), with a variant of stochastic gradient descent recently winning an optimization challenge [22].

Finally, several recent results exploiting *large quantities of labeled data* suggest that with proper initialization and choice of non-linearity, very deep purely supervised networks can be trained successfully without any layerwise pre-training (Ciresan *et al.*, 2010; Glorot *et al.*, 2011a; Krizhevsky *et al.*, 2012). Researchers report than in such conditions, layerwise unsupervised pre-training brought little or no improvement over pure supervised learning from scratch when training for long enough. This reinforces the hypothesis that unsupervised pre-training acts as a prior, which may be less necessary when very large quantities of labeled data are available, but begs the question of why this had not been discovered earlier. The latest results reported in this respect (Krizhevsky *et al.*, 2012) are particularly interesting because they allowed to drastically reduce the error rate of object recognition on a benchmark (the 1000-class ImageNet task) where many more traditional computer vision approaches had been evaluated (`http://www.image-net.org/challenges/LSVRC/2012/results.html`). The main techniques that allowed this success include the following: *efficient GPU training allowing one to train longer* (more than 100 million visits of examples), an aspect first reported by Lee *et al.* (2009a); Ciresan *et al.* (2010), *large number of labeled examples*, *artificially transformed examples* (see Section 11.1), *a large number of tasks* (1000 or 10000 classes for ImageNet), *convolutional architecture* with max-pooling (see section 11 for these latter two techniques), *rectifying non-linearities* (discussed above), *careful initialization* (discussed above), *careful parameter update and adaptive learning rate heuristics*, *layerwise feature normalization* (across features), and a new *dropout* trick based on injecting strong binary multiplicative noise on hidden units. This trick is similar to the binary noise injection used at each layer of a stack of denoising auto-encoder. Future work is hopefully going to help identify which of these elements matter most, how to generalize them across a large variety of tasks and architectures, and in particular contexts where most examples are unlabeled, i.e., including with an unsupervised component in the training criterion.

10.2 Joint Training of Deep Boltzmann Machines

We now consider the problem of joint training of all layers of a specific unsupervised model, the Deep Boltzmann Machine (DBM). Whereas much progress (albeit with many unanswered questions) has been made on jointly training all the layers of deep architectures using back-propagated gradients (i.e., mostly in the supervised setting), much less work has been done on their purely unsupervised counterpart, e.g. with DBMs[23]. Note however that one could hope that the successful techniques described in the previous section could be applied to unsupervised learning algorithms.

Like the RBM, the DBM is another particular subset of the Boltzmann machine family of models where the units are again arranged in layers. However unlike the RBM, the DBM possesses multiple layers of hidden units, with units in odd-numbered layers being conditionally independent given even-numbered layers, and vice-versa. With respect to the Boltzmann energy function of Eq. 7, the DBM corresponds to setting $U = 0$ and a sparse connectivity structure in both V and W. We can make the structure of the DBM more explicit by specifying its energy function. For the model with two hidden layers it is given as:

$$\mathcal{E}_\theta^{\text{DBM}}(v, h^{(1)}, h^{(2)}; \theta) = -\, v^T W h^{(1)} - {h^{(1)}}^T V h^{(2)} - {d^{(1)}}^T h^{(1)} - {d^{(2)}}^T h^{(2)} - b^T v, \quad (29)$$

with $\theta = \{W, V, d^{(1)}, d^{(2)}, b\}$. The DBM can also be characterized as a bipartite graph between two sets of vertices, formed by the units in odd and even-numbered layers (with $v := h^{(0)}$).

10.2.1 *Mean-field approximate inference*

A key point of departure from the RBM is that the posterior distribution over the hidden units (given the visibles) is no longer tractable, due to the interactions between the hidden units. Salakhutdinov and Hinton (2009) resort to a mean-field approximation to the posterior. Specifically, in the case of a model with two hidden layers, we wish to approximate $P\left(h^{(1)}, h^{(2)} \mid v\right)$ with the factored distribution $Q_v(h^{(1)}, h^{(2)}) = \prod_{j=1}^{N_1} Q_v\left(h_j^{(1)}\right) \prod_{i=1}^{N_2} Q_v\left(h_i^{(2)}\right)$, such that the KL divergence $\mathrm{KL}\left(P\left(h^{(1)}, h^{(2)} \mid v\right) \| Q_v(h^1, h^2)\right)$ is minimized or equivalently, that a lower bound to the log likelihood is maximized:

$$\log P(v) > \mathcal{L}(Q_v) \equiv \sum_{h^{(1)}} \sum_{h^{(2)}} Q_v(h^{(1)}, h^{(2)}) \log \left(\frac{P(v, h^{(1)}, h^{(2)})}{Q_v(h^{(1)}, h^{(2)})} \right) \quad (30)$$

Maximizing this lower-bound with respect to the mean-field distribution $Q_v(h^1, h^2)$ (by setting derivatives to zero) yields the following mean field update equations:

$$\hat{h}_i^{(1)} \leftarrow \text{sigmoid}\left(\sum_j W_{ji} v_j + \sum_k V_{ik} \hat{h}_k^{(2)} + d_i^{(1)} \right) \quad (31)$$

$$\hat{h}_k^{(2)} \leftarrow \text{sigmoid}\left(\sum_i V_{ik} \hat{h}_i^{(1)} + d_k^{(2)} \right) \quad (32)$$

Note how the above equations ostensibly look like a *fixed point recurrent neural network*, i.e., with constant input. In the same way that an RBM can be associated with a simple auto-encoder, the above mean-field update equations for the DBM can be associated with a *recurrent auto-encoder*. In that case the training criterion involves the reconstruction error at the last or at consecutive time steps. This type of model has been explored by Savard (2011) and Seung (1998) and shown to do a better job at denoising than ordinary auto-encoders.

Iterating Eq. (31-32) until convergence yields the Q parameters used to estimate the "variational positive phase" of

22. `https://sites.google.com/site/nips2011workshop/optimization-challenges`
23. Joint training of all the layers of a Deep Belief Net is much more challenging because of the much harder inference problem involved.

Eq. 33:

$$\begin{aligned}\mathcal{L}(Q_v) =& \mathbb{E}_{Q_v}\left[\log P(v,h^{(1)},h^{(2)}) - \log Q_v(h^{(1)},h^{(2)})\right]\\ =& \mathbb{E}_{Q_v}\left[-\mathcal{E}_\theta^{\mathrm{DBM}}(v,h^{(1)},h^{(2)}) - \log Q_v(h^{(1)},h^{(2)})\right]\\ &- \log Z_\theta\\ \frac{\partial \mathcal{L}(Q_v)}{\partial\theta} =& -\mathbb{E}_{Q_v}\left[\frac{\partial \mathcal{E}_\theta^{\mathrm{DBM}}(v,h^{(1)},h^{(2)})}{\partial\theta}\right]\\ &+ \mathbb{E}_P\left[\frac{\partial \mathcal{E}_\theta^{\mathrm{DBM}}(v,h^{(1)},h^{(2)})}{\partial\theta}\right] \qquad (33)\end{aligned}$$

Note that this variational learning procedure leaves the "negative phase" untouched. It can thus be estimated through SML or Contrastive Divergence (Hinton, 2000) as in the RBM case.

10.2.2 Training Deep Boltzmann Machines

Despite the intractability of inference in the DBM, its training should not, in theory, be much more complicated than that of the RBM. The major difference being that instead of maximizing the likelihood directly, we instead choose parameters to maximize the lower-bound on the likelihood given in Eq. 30. The SML-based algorithm for maximizing this lower-bound is as follows:

1) Clamp the visible units to a training example.
2) Iterate over Eq. (31-32) until convergence.
3) Generate negative phase samples v^-, $h^{(1)-}$ and $h^{(2)-}$ through SML.
4) Compute $\partial\mathcal{L}(Q_v)/\partial\theta$ using the values obtained in steps 2-3.
5) Finally, update the model parameters with a step of approximate stochastic gradient ascent.

While the above procedure appears to be a simple extension of the highly effective SML scheme for training RBMs, as we demonstrate in Desjardins *et al.* (2012), this procedure seems vulnerable to falling in poor local minima which leave many hidden units effectively dead (not significantly different from its random initialization with small norm).

The failure of the SML joint training strategy was noted by Salakhutdinov and Hinton (2009). As an alternative, they proposed a greedy layer-wise training strategy. This procedure consists in pre-training the layers of the DBM, in much the same way as the Deep Belief Network: i.e. by stacking RBMs and training each layer to independently model the output of the previous layer. A final joint "fine-tuning" is done following the above SML-based procedure.

11 BUILDING-IN INVARIANCE

It is well understood that incorporating prior domain knowledge is an almost sure way to boost performance of any machine-learning-based prediction system. Exploring good strategies for doing so is a very important research avenue. However, if we are to advance our understanding of core machine learning principles, it is important that we keep comparisons between predictors fair and maintain a clear awareness of the prior domain knowledge used by different learning algorithms, especially when comparing their performance on benchmark problems. We have so far tried to present feature learning and deep learning approaches without enlisting specific domain knowledge, only generic inductive biases for high dimensional problems. The approaches as we presented them should thus be potentially applicable to any high dimensional problem. In this section, we briefly review how basic domain knowledge can be leveraged to learn yet better features.

The most prevalent approach to incorporating prior knowledge is to hand-design better features to feed a generic classifier, and has been used extensively in computer vision (e.g. (Lowe, 1999)). Here, we rather focus on how *basic* domain knowledge of the input, in particular its topological structure (e.g. bitmap images having a 2D structure), may be used to *learn* better features.

11.1 Augmenting the dataset with known input deformations

Since machine learning approaches learn from data, it is usually possible to improve results by feeding them a larger quantity of representative data. Thus, a straightforward and generic way to leverage prior domain knowledge of input deformations that are known not to change its class (e.g. small affine transformations of images such as translations, rotations, scaling, shearing), is to use them to generate more training data. While being an old approach (Baird, 1990; Poggio and Vetter, 1992), it has been recently applied with great success in the work of Ciresan *et al.* (2010) who used an efficient GPU implementation ($40\times$ speedup) to train a standard but large deep multilayer Perceptron on deformed MNIST digits. Using both affine and elastic deformations (Simard *et al.*, 2003), with plain old stochastic gradient descent, they reach a record 0.32% classification error rate.

11.2 Convolution and pooling

Another powerful approach is based on even more basic knowledge of merely the *topological structure* of the input dimensions. By this we mean e.g., the 2D layout of pixels in images or audio spectrograms, the 3D structure of videos, the 1D sequential structure of text or of temporal sequences in general. Based on such structure, one can define *local receptive fields* (Hubel and Wiesel, 1959), so that each low-level feature will be computed from only a subset of the input: a neighborhood in the topology (e.g. a sub-image at a given position). This topological locality constraint corresponds to a layer having a very sparse weight matrix with non-zeros only allowed for topologically local connections. Computing the associated matrix products can of course be made much more efficient than having to handle a dense matrix, in addition to the statistical gain from a much smaller number of free parameters. In domains with such topological structure, similar input patterns are likely to appear at different positions, and nearby values (e.g. consecutive frames or nearby pixels) are likely to have stronger dependencies that are also important to model the data. In fact these dependencies can be exploited to *discover* the topology (Le Roux *et al.*, 2008a), i.e. recover a regular grid of pixels out of a set of vectors without any order information, e.g. after the elements have been arbitrarily shuffled in the same way for all examples. Thus a same local feature computation is likely to be relevant at all translated positions of the receptive field. Hence the idea of sweeping such a local feature extractor over the topology: this corresponds to a *convolution*, and transforms an input into a similarly shaped *feature map*. Equivalently to sweeping, this may be seen as static but differently positioned replicated feature extractors that all share the same parameters. This is at the heart of convolutional networks (LeCun *et al.*, 1989, 1998b) which have been applied both to object recognition and to image segmentation (Turaga *et al.*, 2010). Another hallmark of the convolutional architecture is that values computed by the same feature detector applied at several neighboring input locations are then summarized through a pooling operation, typically taking their max or their sum. This confers the resulting pooled feature layer some degree of invariance to input translations, and this style of architecture (alternating selective feature extraction and invariance-creating pooling) has been the basis of convolutional networks, the Neocognitron (Fukushima and Miyake, 1982) and HMAX (Riesenhuber and Poggio, 1999) models, and argued to be the architecture used by mammalian brains for object recognition (Riesenhuber and Poggio, 1999; Serre *et al.*, 2007; DiCarlo *et al.*, 2012). The output of a pooling unit will be the same irrespective of where a specific feature is located inside its pooling region. Empirically the use of pooling seems to contribute significantly to improved classification accuracy in object classification tasks (LeCun *et al.*, 1998b; Boureau *et al.*, 2010, 2011). A successful variant of pooling connected to sparse coding is L2 pooling (Hyvärinen *et al.*, 2009; Kavukcuoglu *et al.*, 2009; Le *et al.*, 2010), for which the pool output is the square root of the possibly weighted sum of squares of filter outputs. Ideally, we would like to generalize feature-pooling so as to *learn what features should be pooled together*, e.g. as successfully done in several papers (Hyvärinen and Hoyer, 2000; Kavukcuoglu *et al.*, 2009; Le *et al.*, 2010; Ranzato and Hinton, 2010; Courville *et al.*, 2011b; Coates and Ng, 2011b; Gregor *et al.*, 2011). In this way, the features learned are becoming *invariant* to the variations captured by the span of the features pooled.

Patch-based training

The simplest approach for learning a convolutional layer in an *unsupervised* fashion is *patch-based training*: simply feeding a generic unsupervised feature learning algorithm with local patches extracted at random positions of the inputs. The resulting feature extractor can then be swiped over the input to produce the convolutional feature maps. That map may be used as a new input for the next layer, and the operation repeated to thus learn and stack several layers. Such an approach was recently used with Independent Subspace Analysis (Le *et al.*, 2011c) on 3D video blocks, reaching the state-of-the-art on Hollywood2, UCF, KTH and YouTube action recognition datasets. Similarly (Coates and Ng, 2011a) compared several feature learners with patch-based training and reached state-of-the-art results on several classification benchmarks. Interestingly, in this work performance was almost as good with very simple k-means clustering as with more sophisticated feature learners. We however conjecture that this is the case only because patches are rather low dimensional (compared to the dimension of a whole image). A large dataset might provide sufficient coverage of the space of e.g. edges prevalent in 6×6 patches, so that a distributed representation is not absolutely necessary. Another plausible explanation for this success is that the clusters identified in each image patch are then *pooled* into a histogram of cluster counts associated with a larger sub-image. Whereas the output of a regular clustering is a one-hot non-distributed code, this histogram is itself a distributed representation, and the “soft” k-means (Coates and Ng, 2011a) representation allows not only the nearest filter but also its neighbors to be active.

Convolutional and tiled-convolutional training

It is also possible to directly train large convolutional layers using an unsupervised criterion. An early approach is that of Jain and Seung (2008) who trained a standard but deep convolutional MLP on the task of denoising images, i.e. as a deep, convolutional, denoising *auto-encoder*. Convolutional versions of the RBM or its extensions have also been developed (Desjardins and Bengio, 2008; Lee *et al.*, 2009a; Taylor *et al.*, 2010) as well as a *probabilistic max-pooling* operation built into Convolutional Deep Networks (Lee *et al.*, 2009a,b; Krizhevsky, 2010). Other unsupervised feature learning approaches that were adapted to the convolutional setting include PSD (Kavukcuoglu *et al.*, 2009, 2010; Jarrett *et al.*, 2009; Farabet *et al.*, 2011; Henaff *et al.*, 2011), a convolutional version of sparse coding called deconvolutional networks (Zeiler *et al.*, 2010), Topographic ICA (Le *et al.*, 2010), and mPoT that Kivinen and Williams (2012) applied to modeling natural textures. Gregor and LeCun (2010a); Le *et al.* (2010) also demonstrated the technique of tiled-convolution, where parameters are shared only between feature extractors whose receptive fields are k steps away (so the ones looking at immediate neighbor locations are not shared). This allows pooling units to be invariant to more than just translations, and is a kind of hybrid between convolutional networks and earlier neural networks with local connections but no weight sharing (LeCun, 1986, 1989).

Alternatives to pooling

Alternatively, one can also use explicit knowledge of the expected invariants expressed mathematically to define transformations that are *robust* to a known family of input deformations, using so-called *scattering operators* (Mallat, 2011; Bruna and Mallat, 2011), which can be computed in a way interestingly analogous to deep convolutional networks and wavelets. Like convolutional networks, the scattering operators alternate two types of operations: convolving with a filter followed by pooling (as a norm). Unlike convolutional networks, the proposed approach keeps at each level all of the information about the input (in a way that can be inverted), and automatically yields a very sparse (but also very high-dimensional) representation. Another difference is that the filters are not learned but instead set so as to guarantee that a priori specified invariances are robustly achieved. Just a few levels were sufficient to achieve impressive results on several benchmark datasets.

11.3 Temporal coherence and slow features

The principle of identifying slowly moving/changing factors in temporal/spatial data has been investigated by many (Becker and Hinton, 1993; Wiskott and Sejnowski, 2002; Hurri and Hyvärinen, 2003; Körding *et al.*, 2004; Cadieu and Olshausen, 2009) as a principle for finding useful representations. In particular this idea has been applied to image sequences and as an explanation for why V1 simple and complex cells behave the way they do. A good overview can be found in Hurri and Hyvärinen (2003); Berkes and Wiskott (2005).

More recently, temporal coherence has been successfully exploited in deep architectures to model video (Mobahi *et al.*, 2009). It was also found that temporal coherence discovered visual features similar to those obtained by ordinary unsupervised feature learning (Bergstra and Bengio, 2009), and a temporal coherence penalty has been *combined* with a training criterion for unsupervised feature learning (Zou *et al.*, 2011), sparse auto-encoders with L1 regularization, in this case, yielding improved classification performance.

The temporal coherence prior can be expressed in several ways. The simplest and most commonly used is just the squared difference between feature values at times t and $t+1$. Other plausible temporal coherence priors include the following. First, instead of penalizing the squared change, penalizing the absolute value (or a similar sparsity penalty) would state that most of the time the change should be exactly 0, which would intuitively make sense for the real-life factors that surround us. Second, one would expect that instead of just being slowly changing, different factors could be associated with their own different time scale. The specificity of their time scale could thus become a hint to disentangle explanatory factors. Third, one would expect that some factors should really be represented by a *group of numbers* (such as x, y, and z position of some object in space and the pose parameters of Hinton *et al.* (2011)) rather than by a single scalar, and that these groups tend to move together. Structured sparsity penalties (Kavukcuoglu *et al.*, 2009; Jenatton *et al.*, 2009; Bach *et al.*, 2011; Gregor *et al.*, 2011) could be used for this purpose.

11.4 Algorithms to Disentangle Factors of Variation

The goal of building invariant features is to remove sensitivity of the representation to directions of variance in the data that are uninformative to the task at hand. However it is often the case that the goal of feature extraction is the *disentangling* or separation of many distinct but informative factors in the data, e.g., in a video of people: subject identity, action performed, subject pose relative to the camera, etc. In this situation, the methods of generating invariant features, such as feature-pooling, may be inadequate.

Roughly speaking, the process of building invariant features can be seen as consisting of two steps (often performed together). First, a set of low-level features are recovered that account for the data. Second, subsets of these low level features are pooled together to form higher-level invariant features, exemplified by the pooling and subsampling layers of convolutional neural networks (Fukushima, 1980; LeCun *et al.*, 1989, 1998c). With this arrangement, the invariant representation formed by the pooling features offers a somewhat incomplete window on the data as the detailed representation of the lower-level features is abstracted away in the pooling procedure. While we would like higher-level features to be more abstract and exhibit greater invariance, we have little control over what information is lost through feature pooling. For example, consider a higher-level feature made invariant to the color of its target stimulus by forming a subspace of low-level features that represent the target stimulus in various colors (forming a basis for the subspace). If this is the only higher-level feature that is associated with these low-level colored features then the color information of the stimulus is lost to the higher-level pooling feature and every layer above. This loss of information becomes a problem when the information that is lost is necessary to successfully complete the task at hand such as object classification. In the above example, color is often a very discriminative feature in object classification tasks. Losing color information through feature-pooling would result in significantly poorer classification performance.

Obviously, what we really would like is for a particular feature set to be invariant to the irrelevant features and disentangle the relevant features. Unfortunately, it is often difficult to determine *a priori* which set of features will ultimately be relevant to the task at hand.

An interesting approach to taking advantage of some of the factors of variation known to exist in the data is the *transforming auto-encoder* (Hinton *et al.*, 2011): instead of a scalar pattern detector (e.g,. corresponding to the probability of presence of a particular form in the input) one can think of the features as organized in groups that include both a pattern detector and *pose parameters* that specify attributes of the detected pattern. In (Hinton *et al.*, 2011), what is assumed a priori is that pairs of inputs (or consecutive inputs) are observed with an *associated value for the corresponding change in the pose parameters*. For example, an animal that controls its eyes *knows* what changes to its ocular motor system were applied when going from one image on its retina to the next image associated with the following saccade and controlled head motion. In that work, it is also assumed that the pose changes are the same for all the pattern detectors, and this makes sense for global changes such as image translation and camera geometry changes. Instead, we would like to *discover* the pose parameters and attributes that should be associated with each feature detector, without having to specify ahead of time what they should be, force them to be the same for all features, and having to necessarily observe the changes in all of the pose parameters or attributes.

The approach taken recently in the Manifold Tangent Classifier, discussed in section 8.3, is interesting in this respect. Without using any supervision or prior knowledge, it finds prominent local *factors of variation* (the tangent vectors to the manifold, extracted from a CAE, interpreted as locally valid input ”deformations”). Higher-level features are subsequently encouraged to be *invariant* to these factors of variation, so that they must depend on other characteristics. In a sense this approach is disentangling valid local deformations along the data manifold from other, more drastic changes, associated to other factors of variation such as those that affect class identity.[24]

One solution to the problem of information loss that would fit within the feature-pooling paradigm, is to consider many overlapping pools of features based on the same low-level feature set. Such a structure would have the potential to learn a redundant set of invariant features that may not cause significant loss of information. However it is not obvious what learning principle could be applied that can ensure that the features are invariant while maintaining as much information as possible. While a Deep Belief Network or a Deep Boltzmann Machine (as discussed in sections 4 and 10.2 respectively) with two hidden layers would, in principle, be able to preserve information into the "pooling" second hidden layer, there is no guarantee that the second layer features are more invariant than the "low-level" first layer features. However, there is some empirical evidence that the second layer of the DBN tends to display more invariance than the first layer (Erhan *et al.*, 2010a). A second issue with this approach is that it could nullify one of the major motivations for pooling features: to reduce the size of the representation. A pooling arrangement with a large number of overlapping pools could lead to as many pooling features as low-level features – a situation that is both computationally and statistically undesirable.

A more principled approach, from the perspective of ensuring a more robust compact feature representation, can be conceived by reconsidering the disentangling of features through the lens of its generative equivalent – feature composition. Since many unsupervised learning algorithms have a generative interpretation (or a way to *reconstruct* inputs from their high-level representation), the generative perspective can provide insight into how to think about disentangling factors. The majority of the models currently used to construct invariant features have the interpretation that their low-level features linearly combine to construct the data.[25] This is a fairly rudimentary form of feature composition with significant limitations. For example, it is not possible to linearly combine a feature with a generic transformation (such as translation) to generate a transformed version of the feature. Nor can we even consider a generic color feature being linearly combined with a gray-scale stimulus pattern to generate a colored pattern. It would seem that if we are to take the notion of disentangling seriously we require a richer interaction of features than that offered by simple linear combinations.

12 Conclusion

This review of representation learning and deep learning has covered three major and apparently disconnected approaches: the probabilistic models (both the directed kind such as sparse coding and the undirected kind such as Boltzmann machines), the reconstruction-based algorithms related to auto-encoders, and the geometrically motivated manifold-learning approaches. Drawing connections between these approaches is currently a very active area of research and is likely to continue to produce models and methods that take advantage of the relative strengths of each paradigm.

Practical Concerns and Guidelines. One of the criticisms addressed to artificial neural networks and deep learning algorithms is that they have many hyper-parameters and variants and that exploring their configurations and architectures is an art. This has motivated an earlier book on the "Tricks of the Trade" (Orr and Muller, 1998) of which LeCun *et al.* (1998a) is still relevant for training deep architectures, in particular what concerns initialization, ill-conditioning and stochastic gradient descent. A good and more modern compendium of good training practice, particularly adapted to training RBMs, is provided in Hinton (2010), while a similar guide oriented more towards deep neural networks can be found in Bengio (2013), both of which are part of a novel version of the above book, entitled "Neural Networks: Tricks of the Trade, Reloaded".

Incorporating Generic AI-level Priors. We have covered many high-level generic priors that we believe could be very useful to bring machine learning closer to AI, and that can be incorporated into representation-learning algorithms. Many of these relate to the assumed existence of multiple underlying factors of variation, whose variations are in some sense orthogonal to each other in input space. They are expected to be organized at multiple levels of abstraction, hence the need for deep architectures, which also have statistical advantages because they allow to *re-use* parameters in a combinatorially efficient way. Only a few of these factors are relevant for any particular example, justifying the sparsity of representation prior. Subsets of these factors explain different random variables of interest (inputs, tasks) and they vary in structured ways in time and space (temporal and spatial coherence). We expect future successful applications of representation learning to refine and increase that list of priors, and to incorporate most of them instead of focusing on only one. Research in training criteria that better take these priors into account are likely to move us closer to the long-term objective of discovering learning algorithms that can *disentangle* the underlying explanatory factors for AI tasks.

Inference. We anticipate that methods based on directly parametrizing a representation function will incorporate more and more of the iterative type of computation one finds in the inference procedures of probabilistic latent-variable models. There is already movement in the other direction, with probabilistic latent-variable models exploiting approximate inference mechanisms that are themselves learned (i.e., producing a parametric description of the representation function). A major appeal of probabilistic models is that the semantics of the latent variables are clear and this allows a clean separation of the problems of modeling (choose the energy function), inference (estimating $P(h|x)$), and learning (optimizing the parameters), using generic tools in each case. On the other hand, doing approximate inference and not taking that approximation into account explicitly in the approximate optimization

24. The changes that affect class identity might, in input space, actually be of similar magnitude to local deformations, but not follow along the manifold, i.e. cross zones of low density.

25. As an aside, if we are given only the values of the higher-level pooling features, we cannot accurately recover the data because we do not know how to apportion credit for the pooling feature values to the lower-level features. This is simply the generative version of the consequences of the loss of information caused by pooling.

for learning could have detrimental effects, hence the appeal of learning approximate inference. More fundamentally, there is the question of the multimodality of the posterior $P(h|x)$. If there are exponentially many probable configurations of values of the factors h_i that can explain x, then we seem to be stuck with very poor inference, either focusing on a single mode (MAP inference), assuming some kind of strong factorization (as in variational inference) or using an MCMC that cannot visit enough modes of $P(h|x)$ to possibly do a good job of inference. What we propose as food for thought is the idea of dropping the requirement of an *explicit* representation of the posterior and settle for an *implicit* representation that exploits potential structure in $P(h|x)$ in order to represent it compactly: even though $P(h|x)$ may have an exponential number of modes, it may be possible to represent it with a small set of numbers. For example, consider computing a deterministic feature representation $f(x)$ that implicitly captures the information about a highly multi-modal $P(h|x)$, in the sense that all the questions (e.g. making some prediction about some target concept) that can be asked from $P(h|x)$ can also be answered from $f(x)$.

Optimization. Much remains to be done to better understand the successes and failures of training deep architectures, both in the supervised case (with many recent successes) and the unsupervised case (where much more work needs to be done). Although regularization effects can be important on small datasets, the effects that persist on very large datasets suggest some optimization issues are involved. Are they more due to local minima (we now know there are huge numbers of them) and the dynamics of the training procedure? Or are they due mostly to ill-conditioning and may be handled by approximate second-order methods? These basic questions remain unanswered and deserve much more study.

Acknowledgments

The author would like to thank David Warde-Farley, Razvan Pascanu and Ian Goodfellow for useful feedback, as well as NSERC, CIFAR and the Canada Research Chairs for funding.

References

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, **10**(2), 251–276.

Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. (2011). Structured sparsity through convex optimization. *CoRR*, **abs/1109.2397**.

Bagnell, J. A. and Bradley, D. M. (2009). Differentiable sparse coding. In *NIPS'2009*, pages 113–120.

Baird, H. (1990). Document image defect models. In *IAPR Workshop, Syntactic & Structural Patt. Rec.*, pages 38–46.

Becker, S. and Hinton, G. (1992). A self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, **355**, 161–163.

Becker, S. and Hinton, G. E. (1993). Learning mixture models of spatial coherence. *Neural Computation*, **5**, 267–277.

Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, **15**(6), 1373–1396.

Bell, A. and Sejnowski, T. J. (1997). The independent components of natural scenes are edge filters. *Vision Research*, **37**, 3327–3338.

Bengio, Y. (1993). A connectionist approach to speech recognition. *International Journal on Pattern Recognition and Artificial Intelligence*, **7**(4), 647–668.

Bengio, Y. (2008). Neural net language models. *Scholarpedia*, **3**(1), 3881.

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 *JMLR W&CP: Proc. Unsupervised and Transfer Learning*.

Bengio, Y. (2013). Practical recommendations for gradient-based training of deep architectures. In K.-R. Müller, G. Montavon, and G. B. Orr, editors, *Neural Networks: Tricks of the Trade, Reloaded*. Springer.

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

Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures. In *ALT'2011*.

Bengio, Y. and LeCun, Y. (2007). 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 *NIPS'2004*, pages 129–136. MIT Press.

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

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *JMLR*, **3**, 1137–1155.

Bengio, Y., Paiement, J.-F., Vincent, P., Delalleau, O., Le Roux, N., and Ouimet, M. (2004). Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In *NIPS'2003*.

Bengio, Y., Delalleau, O., and Le Roux, N. (2006a). The curse of highly variable functions for local kernel machines. In *NIPS'2005*, pages 107–114.

Bengio, Y., Larochelle, H., and Vincent, P. (2006b). Non-local manifold Parzen windows. In *NIPS'2005*, pages 115–122. MIT Press.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *NIPS'2006*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *ICML'09*.

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

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2012a). Better mixing via deep representations. Technical Report arXiv:1207.4404, Universite de Montreal.

Bengio, Y., Alain, G., and Rifai, S. (2012b). Implicit density estimation by local moment matching to sample from auto-encoders. Technical report, arXiv:1207.0057.

Bergstra, J. and Bengio, Y. (2009). Slow, decorrelated features for pretraining complex cell-like networks. In *NIPS'2009*, pages 99–107.

Berkes, P. and Wiskott, L. (2005). Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, **5**(6), 579–602.

Besag, J. (1975). Statistical analysis of non-lattice data. *The Statistician*, **24**(3), 179–195.

Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2012). Joint learning of words and meaning representations for open-text semantic parsing. *AISTATS'2012*.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML'2012*.

Boureau, Y., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in vision algorithms. In *ICML'10*.

Boureau, Y., Le Roux, N., Bach, F., Ponce, J., and LeCun, Y. (2011). Ask the locals: multi-way local pooling for image recognition. In *ICCV'11*.

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

Brand, M. (2003). Charting a manifold. In *NIPS'2002*, pages 961–968. MIT Press.

Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an rbm-derived process. *Neural Computation*, **23**(8), 2053–2073.

Bruna, J. and Mallat, S. (2011). Classification with scattering operators. In *ICPR'2011*.

Cadieu, C. and Olshausen, B. (2009). Learning transformational invariants from natural movies. In *NIPS'2009*, pages 209–216. MIT Press.

Carreira-Perpiñan, M. A. and Hinton, G. E. (2005). On contrastive divergence learning. In *AISTATS'2005*, pages 33–40.

Cayton, L. (2005). Algorithms for manifold learning. Technical Report CS2008-0923, UCSD.

Chen, M., Xu, Z., Winberger, K. Q., and Sha, F. (2012). Marginalized denoising autoencoders for domain adaptation. In *ICML'2012*.

Cho, K., Raiko, T., and Ilin, A. (2010). Parallel tempering is efficient for learning restricted Boltzmann machines. In *IJCNN'2010*.

Cho, K., Raiko, T., and Ilin, A. (2011). Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In *ICML'2011*, pages 105–112.

Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Technical report, arXiv:1202.2745.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, **22**, 1–14.

Coates, A. and Ng, A. Y. (2011a). The importance of encoding versus training with sparse coding and vector quantization. In *ICML'2011*.

Coates, A. and Ng, A. Y. (2011b). Selecting receptive fields in deep networks. In *NIPS'2011*.

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

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, **12**, 2493–2537.

Comon, P. (1994). Independent component analysis - a new concept? *Signal Processing*, **36**, 287–314.

Courville, A., Bergstra, J., and Bengio, Y. (2011a). A spike and slab restricted Boltzmann machine. In *AISTATS'2011*.

Courville, A., Bergstra, J., and Bengio, Y. (2011b). Unsupervised models of images by spike-and-slab RBMs. In *ICML'2011*.

Dahl, G. E., Ranzato, M., Mohamed, A., and Hinton, G. E. (2010). Phone recognition with the mean-covariance restricted Boltzmann machine. In *NIPS'2010*.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, **20**(1), 33–42.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society B*, **39**, 1–38.

Desjardins, G. and Bengio, Y. (2008). Empirical evaluation of convolutional RBMs for vision. Technical Report 1327, Dept. IRO, U. Montréal.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. (2010). Tempered Markov chain monte carlo for training of restricted Boltzmann machine. In *JMLR W&CP: Proc. AISTATS'2010*, volume 9, pages 145–152.

Desjardins, G., Courville, A., and Bengio, Y. (2011). On tracking the partition function. In *NIPS'2011*.

Desjardins, G., Courville, A., and Bengio, Y. (2012). On training deep Boltzmann machines. Technical Report arXiv:1203.4416v1, Université de Montréal.

DiCarlo, J., Zoccolan, D., and Rust, N. (2012). How does the brain solve visual object recognition? *Neuron*.

Donoho, D. L. and Grimes, C. (2003). Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data. Technical Report 2003-08, Dept. Statistics, Stanford University.

Eisner, J. (2012). Learning approximate inference policies for fast prediction. Keynote talk at ICML Workshop on Inferning: Interactions Between Search and Learning.

Erhan, D., Courville, A., and Bengio, Y. (2010a). Understanding representations learned in deep architectures. Technical Report 1355, Université de Montréal/DIRO.

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

Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale fpga-based convolutional networks. In R. Bekkerman, M. Bilenko, and J. Langford, editors, *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge

University Press.

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.

Friedman, J. H. and Stuetzle, W. (1981). Projection pursuit regression. *J. American Statistical Association*, **76**(376), 817–823.

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.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, **15**, 455–469.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS'2010*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011a). Deep sparse rectifier neural networks. In *AISTATS'2011*.

Glorot, X., Bordes, A., and Bengio, Y. (2011b). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML'2011*.

Goodfellow, I., Le, Q., Saxe, A., and Ng, A. (2009). Measuring invariances in deep networks. In *NIPS'2009*, pages 646–654.

Goodfellow, I., Courville, A., and Bengio, Y. (2011). Spike-and-slab sparse coding for unsupervised feature discovery. In *NIPS Workshop on Challenges in Learning Hierarchical Models*.

Goodfellow, I. J., Courville, A., and Bengio, Y. (2012). Spike-and-slab sparse coding for unsupervised feature discovery. arXiv:1201.3382.

Gregor, K. and LeCun, Y. (2010a). Emergence of complex-like cells in a temporal product network with local receptive fields. Technical report, arXiv:1006.0448.

Gregor, K. and LeCun, Y. (2010b). Learning fast approximations of sparse coding. In *ICML'2010*.

Gregor, K., Szlam, A., and LeCun, Y. (2011). Structured sparse coding via lateral inhibition. In *NIPS'2011*.

Gribonval, R. (2011). Should penalized least squares regression be interpreted as Maximum A Posteriori estimation? *IEEE Transactions on Signal Processing*, **59**(5), 2405–2410.

Grosse, R., Raina, R., Kwong, H., and Ng, A. Y. (2007). Shift-invariant sparse coding for audio classification. In *UAI'2007*.

Grubb, A. and Bagnell, J. A. D. (2010). Boosted backpropagation learning for training deep modular networks. In *ICML'2010*.

Gutmann, M. and Hyvarinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS'2010*.

Håstad, J. (1986). Almost optimal lower bounds for small depth circuits. In *STOC'86*, pages 6–20.

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

Henaff, M., Jarrett, K., Kavukcuoglu, K., and LeCun, Y. (2011). Unsupervised learning of sparse features for scalable audio classification. In *ISMIR'11*.

Hinton, G., Krizhevsky, A., and Wang, S. (2011). Transforming auto-encoders. In *ICANN'2011*.

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. (1999). Products of experts. In *ICANN'1999*.

Hinton, G. E. (2000). Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, Gatsby Unit, University College London.

Hinton, G. E. (2010). A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto.

Hinton, G. E. and Roweis, S. (2003). Stochastic neighbor embedding. In *NIPS'2002*.

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 *NIPS'1993*.

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

Hinton, G. E., Osindero, S., Welling, M., and Teh, Y. (2006b). Unsupervised discovery of non-linear structure using contrastive backpropagation. *Cognitive Science*, **30**(4), 725–731.

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

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, **148**, 574–591.

Hurri, J. and Hyvärinen, A. (2003). Temporal coherence, natural image sequences, and the visual cortex. In *NIPS'2002*.

Hyvärinen, A. (2005a). Estimation of non-normalized statistical models using score matching. *J. Machine Learning Res.*, **6**.

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

Hyvärinen, A. (2007). Some extensions of score matching. *Computational Statistics and Data Analysis*, **51**, 2499–2512.

Hyvärinen, A. (2008). Optimal approximation of signal priors. *Neural Computation*, **20**(12), 3087–3110.

Hyvärinen, A. and Hoyer, P. (2000). Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, **12**(7), 1705–1720.

Hyvärinen, A., Karhunen, J., and Oja, E. (2001a). *Independent Component Analysis*. Wiley-Interscience.

Hyvärinen, A., Hoyer, P. O., and Inki, M. (2001b). Topographic independent component analysis. *Neural Computation*, **13**(7), 1527–1558.

Hyvärinen, A., Hurri, J., and Hoyer, P. O. (2009). *Natural Image Statistics: A probabilistic approach to early computational vision*. Springer-Verlag.

Jaeger, H. (2007). Echo state network. *Scholarpedia*, **2**(9), 2330.

Jain, V. and Seung, S. H. (2008). Natural image denoising with convolutional networks. In *NIPS'2008*.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *ICCV'09*.

Jenatton, R., Audibert, J.-Y., and Bach, F. (2009). Structured variable selection with sparsity-inducing norms. Technical report, arXiv:0904.3523.

Jutten, C. and Herault, J. (1991). Blind separation of sources, part I: an adaptive algorithm based on neuromimetic architecture. *Signal Processing*, **24**, 1–10.

Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2008). Fast inference in sparse coding algorithms with applications to object recognition. CBLL-TR-2008-12-01, NYU.

Kavukcuoglu, K., Ranzato, M.-A., Fergus, R., and LeCun, Y. (2009). Learning invariant features through topographic filter maps. In *CVPR'2009*.

Kavukcuoglu, K., Sermanet, P., Boureau, Y.-L., Gregor, K., Mathieu, M., and LeCun, Y. (2010). Learning convolutional feature hierarchies for visual recognition. In *NIPS'2010*.

Kingma, D. and LeCun, Y. (2010). Regularized estimation of image statistics by score matching. In *NIPS'2010*.

Kivinen, J. J. and Williams, C. K. I. (2012). Multiple texture Boltzmann machines. In *AISTATS'2012*.

Körding, K. P., Kayser, C., Einhäuser, W., and König, P. (2004). How are complex cell properties adapted to the statistics of natural stimuli? *J. Neurophysiology*, **91**.

Krizhevsky, A. (2010). Convolutional deep belief networks on cifar-10. Technical report, U. Toronto.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, U. Toronto.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *ICML'2008*.

Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, **10**, 1–40.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR'2006*.

Le, Q., Ngiam, J., Chen, Z., hao Chia, D. J., Koh, P. W., and Ng, A. (2010). Tiled convolutional neural networks. In *NIPS'2010*.

Le, Q., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. (2011a). On optimization methods for deep learning. In *ICML'2011*.

Le, Q. V., Karpenko, A., Ngiam, J., and Ng, A. Y. (2011b). ICA with reconstruction cost for efficient overcomplete feature learning. In *NIPS'2011*.

Le, Q. V., Zou, W. Y., Yeung, S. Y., and Ng, A. Y. (2011c). Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR'2011*.

Le Roux, N., Bengio, Y., Lamblin, P., Joliveau, M., and Kegl, B. (2008a). Learning the 2-D topology of images. In *NIPS'07*.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. (2008b). Topmoumoute online natural gradient algorithm. In *NIPS'07*.

LeCun, Y. (1986). Learning processes in an asymmetric threshold network. In F. Fogelman-Soulié, E. Bienenstock, and G. Weisbuch, editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer-Verlag, Les Houches, France.

LeCun, Y. (1987). *Modèles connexionistes de l'apprentissage*. Ph.D. thesis, Université de Paris VI.

LeCun, Y. (1989). Generalization and network design strategies. In *Connectionism in Perspective*. Elsevier Publishers.

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

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998a). Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998c). Gradient based learning applied to document recognition. *IEEE*, **86**(11), 2278–2324.

Lee, H., Ekanadham, C., and Ng, A. (2008). Sparse deep belief net model for visual area V2. In *NIPS'07*.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009a). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML'2009*.

Lee, H., Pham, P., Largman, Y., and Ng, A. (2009b). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *NIPS'2009*.

Lin, Y., Tong, Z., Zhu, S., and Yu, K. (2010). Deep coding network. In *NIPS'2010*.

Lowe, D. (1999). Object recognition from local scale invariant features. In *ICCV'99*.

Mallat, S. (2011). Group invariant scattering. *Communications in Pure and Applied Mathematics*. to appear.

Marlin, B. and de Freitas, N. (2011). Asymptotic efficiency of deterministic estimators for discrete energy-based models: Ratio matching and pseudolikelihood. In *UAI'2011*.

Marlin, B., Swersky, K., Chen, B., and de Freitas, N. (2010). Inductive principles for restricted Boltzmann machine learning. In *AISTATS'2010*, pages 509–516.

Martens, J. (2010). Deep learning via Hessian-free optimization. In *ICML'2010*, pages 735–742.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *ICML'2011*.

Memisevic, R. and Hinton, G. E. (2010). Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Comp.*, **22**(6).

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 *JMLR W&CP: Proc. Unsupervised and Transfer Learning*, volume 7.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Cernocky, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH'2011*.

Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. In *ICML'2009*.

Mohamed, A., Dahl, G., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Trans. on Audio, Speech and Language Processing*, **20**(1), 14–22.

Montufar, G. F. and Morton, J. (2012). When does a mixture of products contain a product of mixtures? Technical report, arXiv:1206.0387.

Murray, I. and Salakhutdinov, R. (2009). Evaluating probabilities under high-dimensional latent variable models. In *NIPS'2008*, pages 1137–1144.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML'10*.

Narayanan, H. and Mitter, S. (2010). Sample complexity of testing the manifold hypothesis. In *NIPS'2010*.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, **56**, 71–113.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte-Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

Ngiam, J., Chen, Z., Koh, P., and Ng, A. (2011). Learning deep energy models. In *Proc. ICML'2011*. ACM.

Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, **381**, 607–609.

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.

Orr, G. and Muller, K.-R., editors (1998). *Neural networks: tricks of the trade*. Lect. Notes Comp. Sc.,. Springer-Verlag.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, **2**(6).

Poggio, T. and Vetter, T. (1992). Recognition and structure from one 2d model view: Observations on prototypes, object classes and symmetries. AI Lab Memo No. 1347, MIT.

Raiko, T., Valpola, H., and LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *AISTATS'2012*.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data. In *ICML'2007*.

Ranzato, M. and Hinton, G. H. (2010). Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *CVPR'2010*, pages 2551–2558.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In *NIPS'06*.

Ranzato, M., Boureau, Y., and LeCun, Y. (2008). Sparse feature learning for deep belief networks. In *NIPS'2007*.

Ranzato, M., Krizhevsky, A., and Hinton, G. (2010a). Factored 3-way restricted Boltzmann machines for modeling natural images. In *AISTATS'2010*, pages 621–628.

Ranzato, M., Mnih, V., and Hinton, G. (2010b). Generating more realistic images using gated MRF's. In *NIPS'2010*.

Ranzato, M., Susskind, J., Mnih, V., and Hinton, G. (2011). On deep generative models with applications to recognition. In *CVPR'2011*.

Riesenhuber, M. and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*.

Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011a). Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML'2011*.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., and Glorot, X. (2011b). Higher order contractive auto-encoder. In *ECML PKDD*.

Rifai, S., Dauphin, Y., Vincent, P., Bengio, Y., and Muller, X. (2011c). The manifold tangent classifier. In *NIPS'2011*.

Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012). A generative process for sampling contractive auto-encoders. In *ICML'2012*.

Roweis, S. (1997). EM algorithms for PCA and sensible PCA. CNS Technical Report CNS-TR-97-02, Caltech.

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

Salakhutdinov, R. (2010a). Learning deep Boltzmann machines using adaptive MCMC. In *ICML'2010*.

Salakhutdinov, R. (2010b). Learning in Markov random fields using tempered transitions. In *NIPS'2010*.

Salakhutdinov, R. and Hinton, G. E. (2007). Semantic hashing. In *SIGIR'2007*.

Salakhutdinov, R. and Hinton, G. E. (2009). Deep Boltzmann machines. In *AISTATS'2009*, pages 448–455.

Salakhutdinov, R. and Larochelle, H. (2010). Efficient learning of deep Boltzmann machines. In *AISTATS'2010*.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *ICML'2007*, pages 791–798.

Savard, F. (2011). *Réseaux de neurones à relaxation entraînés par critère d'autoencodeur débruitant*. Master's thesis, Université de Montréal.

Schmah, T., Hinton, G. E., Zemel, R., Small, S. L., and Strother, S. (2009). Generative versus discriminative training of RBMs for classification of fMRI images. In *NIPS'2008*, pages 1409–1416.

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

Schwenk, H., Rousseau, A., and Attik, M. (2012). Large, pruned or continuous space language models on a gpu for statistical machine translation. In *Workshop on the future of language modeling for HLT*.

Seide, F., Li, G., and Yu, D. (2011). Conversational speech transcription using context-dependent deep neural networks.

In *Interspeech 2011*, pages 437–440.

Serre, T., Wolf, L., Bileschi, S., and Riesenhuber, M. (2007). Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.*, **29**(3), 411–426.

Seung, S. H. (1998). Learning continuous attractors in recurrent networks. In *NIPS'1997*.

Simard, D., Steinkraus, P. Y., and Platt, J. C. (2003). Best practices for convolutional neural networks. In *ICDAR'2003*.

Simard, P., Victorri, B., LeCun, Y., and Denker, J. (1992). Tangent prop - A formalism for specifying selected invariances in an adaptive network. In *NIPS'1991*.

Simard, P. Y., LeCun, Y., and Denker, J. (1993). Efficient pattern recognition using a new transformation distance. In *NIPS'92*, pages 50–58.

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.

Socher, R., Huang, E. H., Pennington, J., Ng, A. Y., and Manning, C. D. (2011a). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS'2011*.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP'2011*.

Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS'2011*.

Sutskever, I. (2012). *Training Recurrent Neural Networks*. Ph.D. thesis, Departement of computer science, University of Toronto.

Sutskever, I. and Tieleman, T. (2010). On the Convergence Properties of Contrastive Divergence. In *AISTATS'2010*.

Sutskever, I., Hinton, G. E., and Taylor, G. (2009). The recurrent temporal restricted Boltzmann machine. In *NIPS'2008*.

Swersky, K. (2010). *Inductive Principles for Learning Restricted Boltzmann Machines*. Master's thesis, University of British Columbia.

Swersky, K., Ranzato, M., Buchman, D., Marlin, B., and de Freitas, N. (2011). On score matching for energy based models: Generalizing autoencoders and simplifying deep learning. In *Proc. ICML'2011*. ACM.

Taylor, G. and Hinton, G. (2009). Factored conditional restricted Boltzmann machines for modeling motion style. In *ICML'2009*, pages 1025–1032.

Taylor, G., Fergus, R., LeCun, Y., and Bregler, C. (2010). Convolutional learning of spatio-temporal features. In *ECCV'10*, pages 140–153.

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

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *ICML'2008*, pages 1064–1071.

Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. In *ICML'2009*.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal components analysis. *Journal of the Royal Statistical Society B*, **61**(3), 611–622.

Turaga, S. C., Murray, J. F., Jain, V., Roth, F., Helmstaedter, M., Briggman, K., Denk, W., and Seung, H. S. (2010). Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, **22**, 511–538.

van der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *AISTATS'2009*.

van der Maaten, L. and Hinton, G. E. (2008). Visualizing data using t-sne. *J. Machine Learning Res.*, **9**.

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

Vincent, P. and Bengio, Y. (2003). Manifold Parzen windows. In *NIPS'2002*. MIT Press.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*.

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. *J. Machine Learning Res.*, **11**.

Weinberger, K. Q. and Saul, L. K. (2004). Unsupervised learning of image manifolds by semidefinite programming. In *CVPR'2004*, pages 988–995.

Welling, M. (2009). Herding dynamic weights for partially observed random field models. In *UAI'2009*.

Welling, M., Hinton, G. E., and Osindero, S. (2003). Learning sparse topographic representations with products of Student-t distributions. In *NIPS'2002*.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *ICML 2008*.

Weston, J., Bengio, S., and Usunier, N. (2010). Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning*, **81**(1), 21–35.

Wiskott, L. and Sejnowski, T. (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, **14**(4), 715–770.

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

Yu, D., Wang, S., and Deng, L. (2010). Sequential labeling using deep-structured conditional random fields. *IEEE Journal of Selected Topics in Signal Processing*.

Yu, K. and Zhang, T. (2010). Improved local coordinate coding using local tangents. In *ICML'2010*.

Yu, K., Zhang, T., and Gong, Y. (2009). Nonlinear learning using local coordinate coding. In *NIPS'2009*.

Yu, K., Lin, Y., and Lafferty, J. (2011). Learning image representations from the pixel level via hierarchical sparse coding. In *CVPR*.

Yuille, A. L. (2005). The convergence of contrastive divergences. In *NIPS'2004*, pages 1593–1600.

Zeiler, M., Krishnan, D., Taylor, G., and Fergus, R. (2010). Deconvolutional networks. In *CVPR'2010*.

Zou, W. Y., Ng, A. Y., and Yu, K. (2011). Unsupervised learning of visual invariance with temporal coherence. In

NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning.