Deep Learning of Representations for Unsupervised and Transfer Learning

Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Dept. IRO, Université de Montréal. Montréal (QC), H2C 3J7, Canada

Editor: I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver

Abstract

Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features. The objective is to make these higher-level representations more abstract, with their individual features more invariant to most of the variations that are typically present in the training distribution, while collectively preserving as much as possible of the information in the input. Ideally, we would like these representations to disentangle the unknown factors of variation that underlie the training distribution. Such unsupervised learning of representations can be exploited usefully under the hypothesis that the input distribution $P(x)$ is structurally related to some task of interest, say predicting $P(y|x)$. This paper focuses on the context of the Unsupervised and Transfer Learning Challenge, on why unsupervised pre-training of representations can be useful, and how it can be exploited in the transfer learning scenario, where we care about predictions on examples that are not from the same distribution as the training distribution.

Keywords: Deep Learning, unsupervised learning, representation learning, transfer learning, multi-task learning, self-taught learning, domain adaptation, neural networks, Restricted Boltzmann Machines, Auto-encoders.

1. Introduction

Machine learning algorithms attempt to discover structure in data. In their simpler forms, that often means discovering a predictive relationship between variables. More generally, that means discovering where probability mass concentrates in the joint distribution of all the observations. Many researchers have found that the way in which data are represented can make a huge difference in the success of a learning algorithm.

Whereas many practitioners have relied solely on hand-crafting representations, thus exploiting human insights into the problem, there is also a long tradition of learning algorithms that attempt to *discover good representations*. Representation learning is the general context where this paper is situated. What can a good representation buy us? What is a good representation? What training principles might be used to discover good representations?

Supervised machine learning tasks can be abstracted in terms of (X, Y) pairs, where X is an input random variable and Y is a *label* that we wish to predict given X. This paper considers the use of representation learning in the case where *labels for the task of interest are not available at the time of learning the representation*. One wishes to learn the representation either in a purely unsupervised way, or using *labels for other tasks*. This type of setup has been called self-taught learning (Raina et al., 2007) but also falls in the areas of transfer learning, domain adaptation, and multi-task learning (where typically one also has labels for the task of interest) and is related to semi-supervised learning (where one has many unlabeled examples and a few labeled ones).

In order to address this challenge (and *the Unsupervised and Transfer Learning Challenge*[1]), the algorithms described here exploit **Deep Learning**, i.e., learning *multiple levels of representation*. The intent is to discover more *abstract* features in the higher levels of the representation, which hopefully make it easier to *separate from each other the various explanatory factors extent in the data*.

1.1. The Context of The Unsupervised and Transfer Learning Challenge

The challenge was organized according to the following learning setup. The test (and validation) sets have examples from classes not well represented in the training set. They have only a small number of unlabeled examples (4096) and very few labeled examples (1 to 64 per class) available to a Hebbian linear classifier (which discriminates according to the median between the centroids of two classes compared) applied separately to each class against the others. In the second phase of the competition, some labeled examples (from classes other than those in the test or validation sets) are available for the training set. Participants can use the training set (with some irrelevant labels, in the second phase) to construct a representation for test set examples. Typically this is achieved by learning a transformation of the raw input vectors into a new space, which hopefully captures the most important factors of variation present in the unknown generating distribution. That transformation can then be applied to test examples. The challenge server then trains the Hebbian linear classifier on top of that representation, on a small random subset of the test set, and evaluates generalization on the rest of the test set (many random subsets are computed to get an average score). The main difficulties are the following:

1. The input distribution is very different in the test (or validation) set, compared to the training set (for example, the set of classes to be discriminated in the test set may be absent or rare in the training set), making it unclear if anything can be transferred from the training to the test set.

2. Very few labels are available to the linear classifier on the test set, meaning that generalization of the classifier is inherently difficult and sensitive to the particulars of the representation chosen.

3. No labels for the classes of interest (of the test set) are available at all when learning the representation. The labels from the training set might in fact mislead a representation-learning algorithm, because the directions of discrimination which are useful among the training set classes could be useless to discriminate among the test set classes.

This puts great pressure on the representation learning algorithm applied on the training set (unlabeled, in the experiments we performed) to discover really *generic* features likely to be of interest for many classification tasks on such data. Our intuition is that *more abstract*

[1] http://www.causality.inf.ethz.ch/unsupervised-learning.php

features are more likely to fit that stringent requirement, which motivates the use of Deep Learning algorithms.

1.2. Representations as Coordinate Systems

Representation learning is also intimately related to the research in *manifold learning* (Hinton et al., 1997; Tenenbaum et al., 2000; Saul and Roweis, 2002; Belkin and Niyogi, 2003). The objective of manifold learning algorithms is two-fold: identify low-dimensional regions of high-density (called manifold), and construct a coordinate system on these manifolds, i.e., a low-dimensional representation for input examples. Principal Components Analysis (PCA) is the linear ancestor of manifold learning algorithms: it provides a projection of each input vector to a low-dimensional coordinate vector, implicitly defining a low-dimensional hyperplane in input space near which density is hopefully concentrating. The extent of this mass concentration can be measured by the proportion of variance explained by principal eigenvectors of the data covariance matrix. Changes in the directions of the principal components are perfectly captured by the PCA, whereas changes in the orthogonal directions are completely lost. The proportion of the variance in the data captured by the PCA is a good measure of the effectiveness of a PCA dimensionality reduction (for a given choice of number of dimensions). The assumption is that directions where there is very little change in the data do not matter and can be considered noise, but this is not always true, especially when one wrongly assumes that the manifold is linear (e.g., with PCA). Non-linear manifold learning algorithms avoid the linear assumption but retain the notion of a drastic dimensionality reduction.

As we argue more at the end of this paper, although cutting the low-variance directions out (i.e., considering those directions as noise) is often very effective, *it is not always clear what is signal and what is noise*: although the extent of variability is a good hint, it is not perfect. As an example, consider images of faces, and two factors: person identity and pose. Most of the variation in pixel space can be explained by pose (especially the 2-D translation, scaling, and rotation), and two persons of the same sex, age, and hair type will be distinguishable only by looking at low variance components. That is why one often starts by preprocessing such images to align them as much as possible or focus only on images of faces in the same pose, e.g. frontal view.

It is a good thing to test, for one's data, if one can get better classification by simply removing the low-variance components, and in that case one should definitely do it[2]. However, we believe that a more encompassing and more conservative but more ambitious approach is to use a learning algorithm that separates the explanatory factors from each other as much as possible, and let a discriminant classifier pick out those that are relevant to a particular task.

In this context, *overcomplete*[3] *sparse*[4] *representations* have often (Ranzato et al., 2007b, 2008; Goodfellow et al., 2009) been found to work better than dense undercomplete representations (such as produced by PCA). Consider a sparse overcomplete representation in the neighborhood of an input example x. Most local changes around x involve a continuous

[2] and in fact, removing some of the low-variance directions with a preliminary PCA has worked well in the challenge.

[3] overcomplete representation: with more dimensions than the raw input

[4] sparse representation: with many zeros or near-zeros

change of the "active" (non-zero) elements of the representation. Hence the set of active elements of the representation defines a local *chart*, a local coordinate system. Those charts are stitched together through the zero/non-zero transitions that occur when crossing some boundaries in input space. Goodfellow et al. (2009) have found that sparse auto-encoders gave rise to more invariant representations (compared to non-sparse ones), in the sense that a subset of the representation elements (also called features) were more insensitive to input transformations such as translation or rotation of the camera. One advantage of such as an overcomplete representation is that it is not "cramped" in a small-dimensional space. The effective dimensionality (number of non-zeros) can vary depending on where we look. It is very plausible that in some regions of space it may be more appropriate to have more dimensions than in others.

Let $h(x)$ denote the mapping from an input x to its representation $h(x)$. Overcomplete representations which are not necessarily sparse but where h is non-linear are characterized at a particular input point x by a "soft" dimensionality and a "soft" subset of the representation coordinates that are active. The degree to which $h_i(x)$ is active basically depends on $\|\frac{\partial h_i(x)}{\partial x}\|$. When $\|\frac{\partial h_i(x)}{\partial x}\|$ is close to zero, coordinate i is inactive and unresponsive to changes in x, while the active coordinates encode (i.e., respond to) local changes around x. This is what happens with the Contracting Auto-encoder described a bit more in section 3.5.

1.3. Depth

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

The early results on the limitations of shallow circuits regard functions such as the parity function (Yao, 1985), showing that logic gates circuits of depth-2 require exponential size to implement d-bit parity where a deep circuit of depth $O(\log(d))$ could implement it with $O(d)$ nodes. Håstad (1986) then showed that there are functions computable with a polynomial-size logic gate circuit of depth k that require exponential size when restricted to depth $k-1$ (Håstad, 1986). Interestingly, a similar result was proven for the case of circuits made of linear threshold units (formal neurons) (Håstad and Goldmann, 1991), when trying to represent a particular family of functions. A more recent result brings an example of a very large class of functions that cannot be efficiently represented with a small-depth circuit (Braverman, 2011). It is particularly striking that the main theorem regards the representation of functions that capture dependencies in joint distributions. Basically, dependencies that involve more than r variables are difficult to capture by shallow circuits. An r-independent distribution is one that cannot be distinguished from the uniform distribution when looking only at r variables at a time. The proof of the main theorem (which concerns distribution over bit vectors) relies on the fact that order-r polynomials over the reals cannot capture r-independent distributions. The main result is that bounded-depth circuits cannot distinguish data from r-independent distributions from independent noisy bits. We have also recently shown (Delalleau and Bengio, 2011) results for *sum-product networks* (where nodes either compute sums or products, over the reals). We found two families of polynomials that can be efficiently represented with deep circuits, but require exponential size with depth-2 circuits. Interestingly, sum-product networks were recently proposed to efficiently represent high-dimensional joint distributions (Poon and Domingos, 2010).

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

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

In the context of the Unsupervised and Transfer Learning Challenge, the assumption exploited by Deep Learning algorithms goes even further, and is related to the Self-Taught

Learning setup (Raina et al., 2007). In the unsupervised representation-learning phase, one may have access to examples of only some of the classes, and the representation learned should be useful for other classes. One therefore assumes that some of the factors that explain $P(X|Y)$ for Y in the training classes, and that will be captured by the learned representation, will be useful to predict different classes, from the test set. In phase 1 of the competition, only X's from the training classes are observed, while in phase 2 some corresponding labels are observed as well, but no labeled examples from the test set are ever revealed. In our team (LISA), we only used the phase 2 training set labels to help perform model selection, since selecting and fine-tuning features based on their discriminatory ability on training classes greatly increased the risk of removing important information for test classes. See Mesnil et al. (2011) for more details.

2. Greedy Layer-Wise Learning of Representations

The following basic recipe was introduced in 2006 (Hinton and Salakhutdinov, 2006; Hinton et al., 2006; Ranzato et al., 2007a; Bengio et al., 2007):

1. Let $h_0(x) = x$ be the lowest-level representation of the data, given by the observed raw input x.

2. For $\ell = 1$ to L

    Train an unsupervised learning model taking as observed data the training examples $h_{\ell-1}(x)$ represented at level $\ell - 1$, and after training, producing representations $h_\ell(x) = R_\ell(h_{\ell-1}(x))$ at the next level.

From this point on, several variants have been explored in the literature. For supervised learning with fine-tuning, which is the most common variant (Hinton et al., 2006; Ranzato et al., 2007b; Bengio et al., 2007):

3. Initialize a supervised predictor whose first stage is the parametrized representation function $h_L(x)$, followed by a linear or non-linear predictor as the second stage (i.e., taking $h_L(x)$ as input).

4. Fine-tune the supervised predictor with respect to a supervised training criterion, based on a labeled training set of (x, y) pairs, and optimizing the parameters in both the representation stage and the predictor stage.

A supervised variant involves using all the levels of representation as input to the predictor, keeping the representation stage fixed, and optimizing only the predictor parameters (Lee et al., 2009a,b):

3. Train a supervised learner taking as input $(h_k(x), h_{k+1}(x), \ldots, h_L(x))$ for some choice of $0 \leq k \leq L$, using a labeled training set of (x, y) pairs.

A special case of the above is to have $k = L$, i.e., we keep only the top level as input to the classifier without supervised fine-tuning of the representation. Since labels for the test classes are not available (for fine-tuning) in the Unsupervised and Transfer Learning

Challenge, the latter approach makes more sense, but in other settings (especially when the number of labeled examples is large) we have often found fine-tuning to be helpful (Lamblin and Bengio, 2010).

Finally, there is a common unsupervised variant, e.g. for training deep auto-encoders (Hinton and Salakhutdinov, 2006) or a Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009):

3. Initialize an unsupervised model of x based on the parameters of all the stages.

4. Fine-tune the unsupervised model with respect to a global (all-levels) training criterion, based on the training set of examples x.

As detailed in Mesnil et al. (2011), it turned out for the challenge to always work better to have a low-dimensional h_L (i.e. the input to the classifier), e.g., a handful of dimensions. This top-level representation was typically obtained by choosing PCA as the last stage of the hierarchy. In experiments with other kinds of data, with many more labeled examples, we had obtained better results with *high-dimensional* top-level representations (thousands of dimensions). We found higher dimensional top-level representations to be most hurtful for the cases where there are very few labeled examples. Note that the challenge criterion is an average over 1, 2, 4, 8, 16, 32 and 64 labeled examples per class. Because the cases with very few labeled examples were those on which there was most room for improvement (compared to other competitors), it makes sense that the low-dimensional solutions were the most successful.

Another remark is important here. On data sets with a larger labeled training set, we found that the supervised fine-tuning variant (where all the levels are finally tuned with respect to the supervised training criterion) can perform *substantially better* than without supervised fine-tuning (Lamblin and Bengio, 2010).

2.1. Transductive Specialization to Transfer Learning and Domain Adaptation

On the other hand, in the context of the challenge, there were no training labels for the task of interest, i.e., the classes of the test set, so it would not even have been possible to perform meaningful supervised fine-tuning. Worse than that in fact, the *input distribution as well was very different* between the training set and the test set. The large majority of examples from the training set were from classes other than those in the test set. This is a particularly extreme **transfer learning** or **domain adaptation** setup.

How could one hope to generalize in this context? If the training set input distribution had nothing to do with the test set input distribution, then even unsupervised representation-learning on the training set might not be helpful as a learned preprocessing for the test set. The only hope is that a representation-learning algorithm would discover features that capture the generic factors of variation present in all the classes, and that the classifier trained on the test set would then just need to pick up those factors relevant to the discrimination among test set classes. Unfortunately, because of the very small number of labeled examples available to the test set classifier, we found that we could not obtain good results (on the validation set) with high-dimensional representations. This implied that some selection of the relevant features had to be performed even before seeing any label from the test set. We believe that we achieved some of that by using a **transductive strategy**. *The top levels(s) of the unsupervised feature-learning hierarchy were trained purely or mostly on the test set examples.* Since the training set was much larger, we used it to extract a large set of general-purpose features that covered the variations across many classes. The unlabeled test set was then used transductively to *select among the non-linear factors extracted from the training set those few factors varying most in the test set.* Typically this was simply achieved by a simple PCA applied at the last level, trained only on test examples, and with very few leading eigenvectors selected.

3. A Zoo of Possible Layer-Wise Unsupervised Learning Algorithms

3.1. PCA, ICA, Normalization

Existing linear models such as PCA or ICA can be useful as one or more of the levels of a deep hierarchy. In fact, on several of the challenge data sets, we found that using a PCA as **the first and the last level** often worked very well. PCA preserves the global linear directions of maximum variance, and separates them into orthogonal components. The representation learned is the projection on the principal eigenvectors of the input covariance matrix. This corresponds to a coordinate system associated with a linear manifold spanned by these eigenvectors (centered at the center of mass of the data). For the first PCA layer, we typically kept a fairly large number of directions, so the main effect of that step is to smooth the input distribution by eliminating some of the variations involved in the least globally varying directions. Optionally, the PCA transformation can include a whitening step which means that the projections are normalized to variance 1 (by dividing each projection by the square root of the corresponding eigenvalue, i.e., component variance).

Whereas PCA can already perform a kind of normalization across examples (by subtracting the mean over examples and dividing by the standard deviation over examples, in the chosen directions), there is a complementary form of normalization which we have found useful. It is a simple variant of the *contrast normalization* commonly employed as an intermediate step in deep convolutional neural networks (LeCun et al., 1989, 1998a). The idea is to normalize across the elements of each input vector, by subtracting the mean and dividing by the standard deviation across elements of the input vector.

3.2. Auto-encoders

An auto-encoder defines a reconstruction $r(x) = g(h(x))$ of the input x from the composition of an *encoder* $h(\cdot)$ and a *decoder* $g(\cdot)$. In general both are parametrized and the most common parametrization corresponds to $r(x)$ being the output of a one-hidden layer neural network (taking x as input) and $h(x)$ being the non-linear output of the hidden layer. Training proceeds by minimizing the average of reconstruction errors, $L(r(x), x)$. If the encoder and decoder are linear and $L(r(x), x) = ||r(x) - x||^2$ is the square error, then $h(x)$ learns to span the principal eigenvectors of the input, i.e., being equivalent (up to a rotation) to a PCA (Bourlard and Kamp, 1988). However, with a non-linear encoder, one obtains a representation that can be greedily stacked and often yields better representations with deeper encoders (Bengio et al., 2007; Goodfellow et al., 2009). A probabilistic interpretation of reconstruction error is simply as a particular form of *energy function* (Ranzato et al., 2008) (the logarithm of an unnormalized probability density function). It means that examples with low reconstruction error have higher probability according to the model. A sparsity term in the energy function has been used to allow overcomplete representations (Ranzato et al., 2007b, 2008) and shown to yield features that are (for some of them) more invariant to geometric transformations of images (Goodfellow et al., 2009). A successful alternative (Bengio et al., 2007; Vincent et al., 2008) to the square reconstruction error in the case of inputs that are binary or in the (0,1) interval (like pixel intensities) is the sum of KL divergences between the binomial probability distributions associated with each input x_i and with each reconstruction $r_i(x)$ (both seen as probabilities for a binary event).

3.3. RBMs

As shown in Bengio and Delalleau (2009), the reconstruction error gradient for auto-encoders can be seen as an approximation of the Contrastive Divergence (Hinton, 1999, 2002) update rule for Restricted Boltzmann Machines (Hinton et al., 2006). Boltzmann Machines are undirected graphical models, defined by an energy function which is related to the joint probability of inputs (visible) x and hidden (latent) variables h through

$$P(x,h) = e^{-\text{energy}(x,h)}/Z$$

where the normalization constant Z is called the partition function, and the marginal probability of the observed data (which is what we want to maximize) is simply $P(x) = \sum_h P(x,h)$ (summing or integrating over all possible configurations of h). A Boltzmann Machine is the equivalent for binary random variables of the multivariate Gaussian distribution for continuous random variables, in the sense that it is defined by an energy function that is a second-order polynomial in the random bit values. Both are particular kinds of Markov Random Fields (undirected graphical models), but the partition function of the Boltzmann Machine is intractable, which means that approximations of the log-likelihood gradient $\frac{\partial \log P(x)}{\partial \theta}$ must be employed to train it (where θ is the set of parameters).

Restricted Boltzmann Machines (RBMs) are Boltzmann Machines with a restriction in the connection pattern of the graphical model between variables, forming a bipartite graph with two groups of variables: the input (or visible) and latent (or hidden) variables. Whereas the original RBM employs binomial hidden and visible units, which worked well on data such as MNIST (where grey levels actually correspond to probabilities of turning on a pixel), the original extension of RBMs to continuous data (the Gaussian RBM) has not been as successful as more recent continuous-data RBMs such as the mcRBM (Ranzato and Hinton, 2010), the mPoT model (Ranzato et al., 2010) and the **spike-and-slab RBM** (Courville et al., 2011), which was used in the challenge. The spike-and-slab RBM energy function allows hidden units to either push variance up or down in different directions, and it can be efficiently trained thanks to a 3-way block Gibbs sampling procedure.

RBMs are defined by their energy function, and when it is tractable (which is usually the case), their *free energy* function is:

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

The log-likelihood gradient can be defined in terms of the gradient of the free energy on observed (so-called positive) data samples x and on (so-called negative) model samples

$\tilde{x} \sim P(\tilde{x})$:

$$-\frac{\partial \log P(x)}{\partial \theta} = \frac{\partial \text{FreeEnergy}(x)}{\partial \theta} - E[\frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta}]$$

where the expectation is over $\tilde{x} \sim P(\tilde{x})$. When the free energy is tractable, the first term can be computed readily, whereas the second term involves sampling from the model. Various kinds of RBMs can be trained by approximate maximum likelihood stochastic gradient descent, often involving a Monte-Carlo Markov Chain to obtain those model samples. See Bengio (2009) for a much more complete tutorial on this subject, along with Hinton (2010) for tips and tricks.

3.4. Denoising Auto-encoders

Denoising auto-encoders training is a simple variation on auto-encoder training: try to reconstruct the clean original input from an artificially and stochastically corrupted version of it, by minimizing the denoising reconstruction error. Denoising auto-encoders are simply trained by stochastic gradient descent, typically using mini-batches (of 20 to 200 examples) in order to take advantage of faster matrix-matrix operations on CPUs or GPUs. Denoising auto-encoders solve one of the thorny limitations of ordinary auto-encoders: the representation can be overcomplete without causing any problem of learning a trivial identity function. More hidden units just means that the model can more finely represent the input distribution. Denoising auto-encoders have recently been shown to be directly related to score matching (Vincent, 2011), an induction principle that can replace maximum likelihood when it is not tractable (and the inputs are continuous-valued). The score matching criterion is the squared norm of the difference between the model's *score* (gradient $\frac{\partial \log P(x)}{\partial x}$ of the log-likelihood with respect to the input x) and the score of the true data generating density (which is unknown, but from which we have samples). A simple way to understand the connection between denoising auto-encoders and score matching is the following. Considering that reconstruction error is an energy function, the reconstruction from an auto-encoder normally goes from a lower-probability (higher energy) input configuration to a nearby higher-probability (lower energy) one, so the difference $r(\tilde{x}) - \tilde{x}$ between reconstruction and input is the model's view of a direction of maximum increase in probability (i.e., the model's score). On the other hand, when one takes a training sample x and one randomly corrupts it into $\tilde{x}$, one typically obtains a lower probability neighbor, i.e., the vector $x - \tilde{x}$ is nature's hint about a direction of rapid increase in probability (when starting at $\tilde{x}$). The squared difference of these two differences is just the denoising reconstruction error $(r(\tilde{x}) - x)^2$, in the case of the squared error reconstruction loss.

In the challenge, we used (Mesnil et al., 2011) particular denoising auto-encoders that are well suited for data with sparse high-dimensional inputs. Instead of the usual sigmoid or tanh non-linear hidden unit activations, these auto-encoders are based on *rectifier* units ($\max(x, 0)$ instead of tanh) with L1 penalty in the training criterion, which tends to make the hidden representation sparse. Stochastic rectifier units had been introduced in the context of RBMs earlier (Nair and Hinton, 2010) and we have found them to be extremely useful for deterministic deep networks (Glorot et al., 2011a) and denoising auto-encoders (Glorot et al., 2011b). A recent extension of denoising auto-encoders is particularly useful for two of the challenge data sets in which the input vectors are very large and sparse. It addresses a particularly troubling issue when training auto-encoders on large sparse vectors: whereas the encoder can take advantage of the numerous zeros in the input vector (it does not need to do any computation for them), the decoder needs to make reconstruction predictions and compute reconstruction error for all the inputs, including the zeros. With the sampled reconstruction algorithm (Dauphin et al., 2011), one only needs to compute reconstructions and reconstruction error for a small stochastically selected subset of the zeros, yielding very substantial speed-ups (20-fold in the experiments of Dauphin et al. (2011)), the more so as the fraction of non-zeros decreases.

3.5. Contractive Auto-encoders

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

4. Tricks and Tips

A good starting point for tricks and tips relevant to training deep architectures, and in particular Restricted Boltzmann Machines (RBMs), is Hinton (2010). An older guide which is also useful to some extent is Orr and Muller (1998), and in particular LeCun et al. (1998b), since many of the ideas from neural networks training can be exploited here. A more recent guide for training the kinds of neural networks described here can be found in Bengio (2013).

4.1. Monitoring Performance During Training

RBMs are tricky because although there are good estimators of the log-likelihood *gradient*, there are no known cheap ways of estimating the log-likelihood itself (Annealed Importance Sampling (Murray and Salakhutdinov, 2009) is an expensive way of doing it). A poor man's option is to measure reconstruction error (as if the parameters were those of an auto-encoder), which works well for the beginning of training but does not help to choose a stopping point (e.g. to avoid overfitting). The practical solution is to save the model weights at different numbers of epochs (e.g., 5, 10, 20, 50) and plug the learned representation into a supervised classifier (for each of these training durations) in order to decide what training duration to select.

In the case of denoising auto-encoders, on the other hand, the denoising reconstruction error is a good measure of the model's progress (since it corresponds to the training criterion) and it can be used for early stopping. However, the best generative model or the one with the best denoising is not always the one that works best in terms of providing a good representation for a classifier. This is especially true in the transfer setting of the competition, where the training distribution is different from the test and validation distributions. In that case, the expensive solution of evaluating validation classification error at different training durations is the approach we have chosen. The training criterion of contractive auto-encoders can also be used as a good monitoring device (and its value on a validation set used to perform early stopping). Note that we did not really "stop" training, we only recorded representations at different points in the training trajectory and estimated Area under the Learning Curve (ALC[5]) or other criteria associated with each. The advantage is that we do not need to retrain a separate model from scratch for each of the possible durations tested.

4.2. Random Search and Greedy Layer-wise Strategy

Because one can have a different type of representation-learning model at each layer, and because each of these learning algorithms has several hyper-parameters, there is a huge number of possible configurations and choices one can make in exploring the kind of deep architectures that led to the winning entry of the challenge. There are two approaches that practitioners of machine learning typically employ to deal with hyper-parameters. One is manual trial and error, i.e., a human-guided search. The other is a grid search, i.e., choosing a set of values for each hyper-parameter and training and evaluating a model for each combination of values for all the hyper-parameters. Both work well when the number of hyper-parameters is small (e.g. 2 or 3) but break down when there are many more[6]. More systematic approaches are needed. An approach that we have found to scale better is based on random search and greedy exploration. The idea of random search (Bergstra and Bengio, 2011, 2012) is simple and can advantageously replace grid search. Instead of forming a regular grid by choosing a small set of values for each hyper-parameter, one

[5] The ALC is the sum of the accuracy measure for different number of labeled training examples. The accuracy measure used here is the AUC or Area Under the Curve. See `http://www.causality.inf.ethz.ch/ul_data/DatasetsUTLChallenge.pdf`

[6] Experts can handle many hyper-parameters, but results become less reproducible and algorithms less accessible to non-experts.

defines a distribution from which to sample values for each hyper-parameter, e.g., the log of the learning rate could be taken as uniform between $\log(0.1)$ and $\log(10^{-6})$, or the log of the number of hidden units or principal components could be taken as uniform between $\log(2)$ and $\log(5000)$. The main advantage of random (or quasi-random) search over a grid is that when some hyper-parameters have little or no influence, random search does not waste any computation, whereas grid search will redo an exponential number of experiments (with respect to number of hyper-parameters) that are equivalent and do not bring any new information (because many of them have the same value for hyper-parameters that matter and different values for hyper-parameters that do not). Instead, with random search, every experiment is different, thus bringing more information. In addition, random search is convenient because even if some jobs are not finished, one can draw conclusions from the jobs that are finished. In fact, one can use the results on subsets of the experiments to establish confidence intervals (the experiments are now all iid), and draw a curve (with confidence interval) showing how performance improves as we do more exploration. Of course, it would be even better to perform a sequential optimization (Bergstra et al., 2011) (such as Bayesian Optimization (Brochu et al., 2009)) in order to take advantage of results of training experiments as they are obtained and sample in more promising regions of configuration space, but more research needs to be done towards this. On the other hand, random search is very easy and does not introduce hyper-hyper-parameters.

Another trick that we have used successfully in the past and in the challenge is the idea of a greedy search. Since the deep architecture is obtained by stacking layers and each layer comes with its own choices and hyper-parameters, the general strategy is the following. First optimize the choices for the first layer (e.g., try to find which single-layer learning algorithm and its hyper-parameters give best results according to some criterion such as validation set classification error or ALC). Then keep that best choice (or a few of the best choices found) and explore choices for the second layer, keeping only the best overall choice (or a few of the best choices found) among the 2-layer systems tested. This procedure can then be continued to add more layers, without the computational cost exploding with the number of layers (it just grows linearly).

4.3. Hyper-Parameters

The single most important hyper-parameter for most of the algorithms described here is the learning rate. A too small learning rate means slow convergence, or convergence to a poor performance given a finite budget of computation time. A too large learning rate gives poor results because the training criterion may increase or oscillate. The optimal learning rate for one data set and architecture may be too large or too small when changing one or the other, so it is worth optimizing the learning rate. Like most numerical hyper-parameters, the learning rates should be explored in the log-domain, and there is not much to be gained by refining it more than a factor of 2, whereas the dynamic range explored could be around 10^6, learning rates are typically below 1. To efficiently search for a good learning rate, a greedy heuristic that we used is based on the following strategy. Start with a large learning rate and reduce it (by a factor 3) until training does not diverge. The largest learning rate which does not give divergent training (increasing training error) is usually a very good choice of learning rate.

For the challenge, another very sensitive hyper-parameter is the number of dimensions of the top-level representation fed to the classifier. It should probably be close to or related to the true number of classes (more classes would require more dimensions to be separated easily by a linear classifier).

Early stopping is another handy trick to speed-up model search, since it can be used to detect overfitting (even in the unsupervised learning sense) for a low computational cost. After each training iteration one can compute an indicator of generalization error (either from the application of the unsupervised learning criterion on the validation set or even by training a linear classifier on a pseudo-validation set, as described below, sec. 4.5).

4.4. Visualization

Since the validation set ALC was an unreliable indicator of test set ALC, we used several strategies in the second phase of the competition to help guide the model selection. One of them is simply visualization of the representations as cloud points. One can visualize 3 dimensions at a time, either the leading 3 or a subset of the leading ones. To order and select dimensions we used PCA or t-SNE dimensionality reduction (van der Maaten and Hinton, 2008).

4.5. Simulating the Final Evaluation Scenario

Another strategy that often comes handy is to simulate (as much as possible) the final evaluation scenario, even in the absence of the test labels. In the case of the second phase of the competition, some of the training classes labels are available. Thus we could simulate the final evaluation scenario by choosing a subset of the training classes as "pseudo training set" and the rest as "pseudo test set", doing unsupervised training on the pseudo training set (or the union of pseudo training and pseudo test sets) and training the linear classifier using the pseudo test set. We then considered hyper-parameter settings that led not only to high accuracy in average across different choices of class subsets (for the pseudo train/test split), but also to high robustness (low variance) across these splits.

5. Examples in Transfer Learning

Deep Learning seems well suited to transfer learning because it focuses on learning representations and in particular "abstract" representations, representations that ideally disentangle the factors of variation present in the input.

This has been demonstrated already not only in this competition (see Mesnil et al. (2011)) for more details), but also in several other instances. For example, in Bengio et al. (2011), it has been shown that a deep learner can take more advantage of out-of-distribution training examples (whose distribution differs from the test distribution) than a shallow learner. Two settings were explored, with both results illustrated in Figure 1. In the first one (left hand side of figure), training examples (character images) are distorted by adding all kinds of noises and random transformations (coherent with character images), but the target distribution contains clean examples. Training on the distorted examples was found to help the deep learners (SDA{1,2}) more than the shallow ones (MLP{1,2}), when the goal is to test on the clean examples. In the second setting (right hand side of figure), i.e., the multi-task setting, training is on all 62 classes available, but the target distribution of interest is a restriction to a subset of the classes.

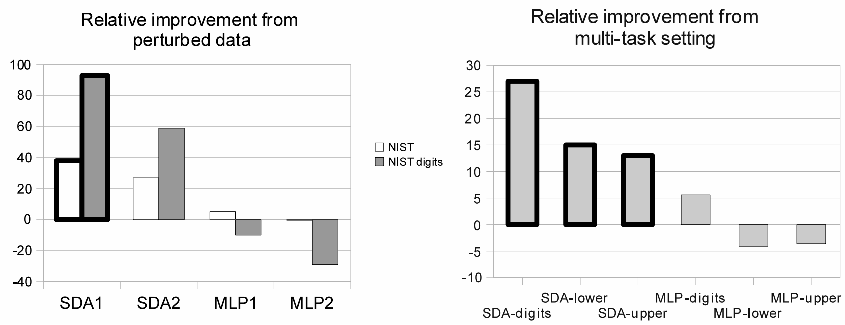

Figure 1: Relative improvement in character classification error rate due to out-of-distribution examples. Left: Improvement (or loss, when negative) induced by out-of-distribution examples (perturbed data). Right: Improvement (or loss, when negative) induced by multi-task learning (training on all character classes and testing only on either digits, upper case, or lower-case). The deep learner (stacked denoising auto-encoder) benefits more from out-of-distribution examples, compared to a shallow MLP. {SDA,MLP}1 and {SDA,MLP}2 are respectively trained on different types of distortions. The NIST set includes all 62 character classes while NIST digits include only the 10 digits. Reproduced from Bengio et al. (2011).

Another successful transfer example also using stacked denoising auto-encoders arises in the context of *domain adaptation*, i.e., where one trains an unsupervised representation based on examples from a set of domains but a classifier is then trained from few examples of only one domain. In that case, unlike in the challenge, the output variable always has the same semantics, but the input distribution (and to a lesser extent the relation between input and output) changes from domain to domain. Glorot et al. (2011b) applied stacked denoising auto-encoders with sparse rectifiers (the same as used for the challenge) to domain adaptation in *sentiment analysis* (predicting whether a user liked a disliked a product based on a short review). Some of the results are summarized in Figure 2, comparing transfer ratio, which indicates relative error when testing in-domain vs out-of-domain, i.e., how well transfer works (see Glorot et al. (2011b) for more details). The stacked denoising auto-encoders (SDA) are compared with the state of the art methods: SCL (Blitzer et al., 2006) or Structural Correspondence Learning, MCT (Li and Zong, 2008) or Multi-label Consensus Training, SFA (Pan et al., 2010) or Spectral Feature Alignment, and T-SVM (Sindhwani and Keerthi, 2006) or Transductive SVM. The SDA$_{\text{sh}}$ (Stacked Denoising Auto-encoder trained on all domains) clearly beats the state-of-the-art.

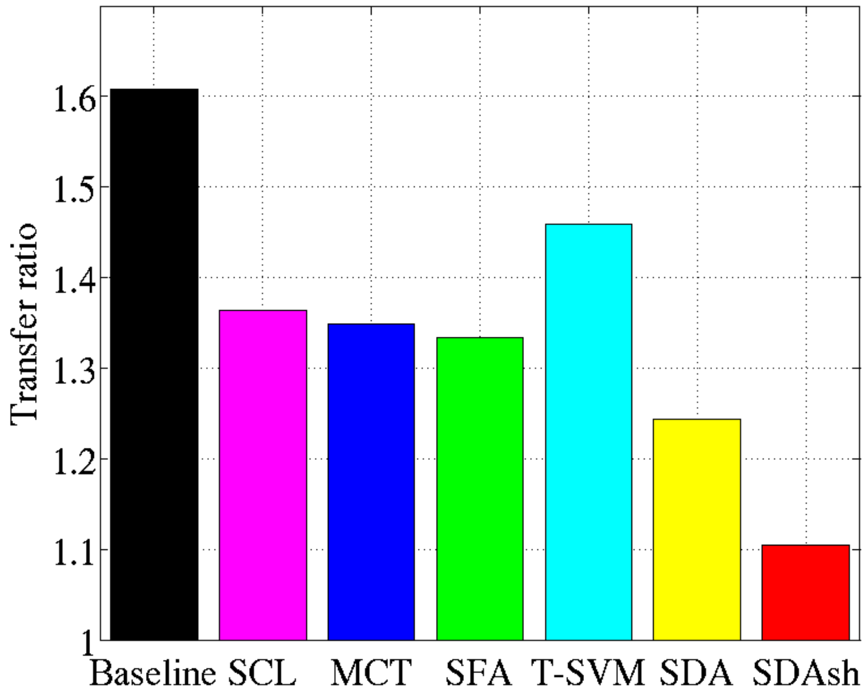

Figure 2: **Transfer ratios** on the Amazon benchmark. Both SDA-based systems outperforms the rest, and SDA$_{sh}$ (unsupervised training on all domains) is best. Reproduced from Glorot et al. (2011b).

6. Moving Forward: Disentangling Factors of Variation

In spite of all the nice results described here, and in spite of winning the final evaluation of the challenge, it is clear to the author that research in Deep Learning of representations is only in its infancy, and that much more should be done to improve the learning algorithms. In particular, it is the author's belief that these algorithms would be much more useful in transfer learning if they could better *disentangle the underlying factors of variation.* In a sense it is obvious that if we had algorithms that could do that really well, than most learning tasks (supervised learning, transfer learning, reinforcement learning, etc.) would become much easier, and the effect would be most felt when only very few labeled examples for the final task of interest are present. The question is whether we can actually improve in this direction, and the hypothesis we propose is that by explicitly designing the models and training criterion towards that objective, there is much to be gained. Ultimately, one can view the problem of learning from very few labeled examples of the task of interest almost as a an inference problem ("given that I define a new class based on this particular example, what is the probability that this other example also belongs to it?"), where the parameters of the model (i.e., the representation) have already been established through prior training on many more related examples (labeled or not) which help to capture the underlying factors of variation, some of which are relevant in the target task.

Acknowledgments

The author wants to thank NSERC, the Canada Research Chairs, mPrime, Compute Canada and FQRNT for their support, and the other authors of the companion paper (Mesnil et al., 2011) for their contributions to many of the results and ideas illustrated here.

References

Mikhail Belkin and Partha Niyogi. Using manifold structure for partially labeled classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15 (NIPS'02)*, Cambridge, MA, 2003. MIT Press.

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

Yoshua Bengio. 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, 2013.

Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, June 2009.

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

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press, 2007.

Yoshua Bengio, Frédéric Bastien, Arnaud Bergeron, Nicolas Boulanger-Lewandowski, Thomas Breuel, Youssouf Chherawala, Moustapha Cisse, Myriam Côté, Dumitru Erhan, Jeremy Eustache, Xavier Glorot, Xavier Muller, Sylvain Pannetier Lebeuf, Razvan Pascanu, Salah Rifai, François Savard, and Guillaume Sicard. Deep learners benefit more from out-of-distribution examples. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization, 2011. *The Learning Workshop*, Fort Lauderdale, Florida.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

James Bergstra, Rémy Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS'2011*, 2011.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP '06*, 2006.

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294, 1988.

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

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report TR-2009-23, Department of Computer Science, University of British Columbia, November 2009.

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

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

Aaron Courville, James Bergstra, and Yoshua Bengio. Unsupervised models of images by spike-and-slab RBMs. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*, June 2011.

Yann Dauphin, Xavier Glorot, and Yoshua Bengio. Sampled reconstruction for large-scale learning of embeddings. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*, June 2011.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, 2011.

Xavier Glorot, Antoire Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011a.

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

Ian Goodfellow, Quoc Le, Andrew Saxe, and Andrew Ng. Measuring invariances in deep networks. In Yoshua Bengio, Dale Schuurmans, Christopher Williams, John Lafferty, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 646–654, 2009.

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

Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

G. E. Hinton, P. Dayan, and M. Revow. Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8:65–74, 1997.

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

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

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

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

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Pascal Lamblin and Yoshua Bengio. Important gains from supervised fine-tuning of deep architectures on large labeled sets. NIPS*2010 Deep Learning and Unsupervised Feature Learning Workshop, 2010.

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

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998b.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Léon Bottou and Michael Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, Montreal (Qc), Canada, 2009a.

Honglak Lee, Peter Pham, Yan Largman, and Andrew Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Yoshua Bengio, Dale Schuurmans, Christopher Williams, John Lafferty, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 1096–1104, 2009b.

Shoushan Li and Chengqing Zong. Multi-domain adaptation for sentiment classification: Using multiple classifier combining methods. In *Proc. of NLP-KE '08*, 2008.

Grégoire Mesnil, Yann Dauphin, Xavier Glorot, Salah Rifai, Yoshua Bengio, Ian Goodfellow, Erick Lavoie, Xavier Muller, Guillaume Desjardins, David Warde-Farley, Pascal Vincent, Aaron Courville, and James Bergstra. Unsupervised and transfer learning challenge: a deep learning approach. In Isabelle Guyon, G. Dror, V Lemaire, G. Taylor, and D. Silver, editors, *JMLR W& CP: Proceedings of the Unsupervised and Transfer Learning challenge and workshop*, volume 7, 2011.

Iain Murray and Ruslan Salakhutdinov. Evaluating probabilities under high-dimensional latent variable models. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Leon Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, volume 21, pages 1137–1144, 2009.

V. Nair and G. E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML '10*, 2010.

Genevieve Orr and Klaus-Robert Muller, editors. *Neural networks: tricks of the trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1998. ISBN 3-540-65311-2 (paperback).

Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proc. of WWW '10*, 2010.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, Whistler, Canada, 2010.

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In Zoubin Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 759–766. ACM, 2007.

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

M. Ranzato, V. Mnih, and G. Hinton. Generating more realistic images using gated MRF's. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23 (NIPS'10)*, pages 2002–2010, 2010.

M.A. Ranzato and G.E. Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE, 2010.

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

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

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contracting auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*, June 2011.

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

L. Saul and S. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2002.

Vikas Sindhwani and S. Sathiya Keerthi. Large scale semi-supervised linear svms. In *Proc. of SIGIR '06*, 2006.

Joshua Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, November 2008. URL http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, to appear, 2011.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 1096–1103. Omnipress, 2008.

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