# An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation

**Hugo Larochelle** LAROCHEH@IRO.UMONTREAL.CA
**Dumitru Erhan** ERHANDUM@IRO.UMONTREAL.CA
**Aaron Courville** COURVILA@IRO.UMONTREAL.CA
**James Bergstra** BERGSTRJ@IRO.UMONTREAL.CA
**Yoshua Bengio** BENGIOY@IRO.UMONTREAL.CA
Dept. IRO, Université de Montréal C.P. 6128, Montreal, Qc, H3C 3J7, Canada

## Abstract

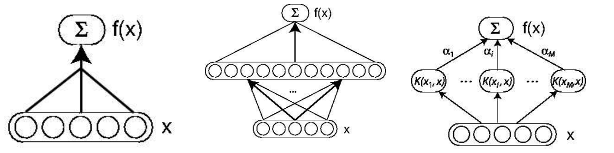
Recently, several learning algorithms relying on models with deep architectures have been proposed. Though they have demonstrated impressive performance, to date, they have only been evaluated on relatively simple problems such as digit recognition in a controlled environment, for which many machine learning algorithms already report reasonable results. Here, we present a series of experiments which indicate that these models show promise in solving harder learning problems that exhibit many factors of variation. These models are compared with well-established algorithms such as Support Vector Machines and single hidden-layer feed-forward neural networks.

## 1. Introduction

Several recent empirical and theoretical results have brought deep architectures to the attention of the machine learning community: they have been used, with good results, for dimensionality reduction (Hinton & Salakhutdinov, 2006; Salakhutdinov & Hinton, 2007), and classification of digits from the MNIST data set (Hinton et al., 2006; Bengio et al., 2007). A core contribution of this body of work is the training strategy for a family of computational models that is similar or identical to traditional multilayer perceptrons with sigmoidal hidden units. Traditional gradient-based optimization strategies are not effective when the gradient must be propagated across multiple non-linearities. Hinton (2006) gives empirical evidence that a sequential, greedy, optimization of the weights of each layer using the generative training criterion of a Restricted Boltzmann Machine tends to initialize the weights such that global gradient-based optimization can work. Bengio et al. (2007) showed that this procedure also worked using the autoassociator unsupervised training criterion and empirically studied the sequential, greedy layer-wise strategy. However, to date, the only empirical comparison on classification problems between these deep training algorithms and the state-of-the-art has been on MNIST, on which many algorithms are relatively successful and in which the classes are known to be well separated in the input space. It remains to be seen whether the advantages seen in the MNIST dataset are observed in other more challenging tasks.

Ultimately, we would like algorithms with the capacity to capture the complex structure found in language and vision tasks. These problems are characterized by many factors of variation that interact in nonlinear ways and make learning difficult. For example, the NORB dataset introduced by LeCun et al. (2004) features toys in real scenes, in various lighting, orientation, clutter, and degrees of occlusion. In that work, they demonstrate that existing general algorithms (Gaussian SVMs) perform poorly. In this work, we propose a suite of datasets that spans some of the territory between MNIST and NORB–starting with MNIST, and introducing multiple factors of variation such as rotation and background manipulations. These toy datasets allow us to test the limits of current state-of-the-art algorithms, and explore the behavior of the newer deep-architecture training procedures, *with architectures not tailored to machine vision*. In a very limited but significant way, we believe that these problems are closer to "real world" tasks, and can serve as milestones on the road to AI.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).



(a) Linear model architecture (b) Single layer neural network architecture (c) Kernel SVM architecture

*Figure 1.* Examples of models with shallow architectures.

### 1.1. Shallow and Deep Architectures

We define a *shallow model* as a model with very few layers of composition, e.g. linear models, one-hidden-layer neural networks and kernel SVMs (see figure 1). On the other hand, *deep architecture models* are such that their output is the result of the composition of some number of computational units, commensurate with the amount of data one can possibly collect, i.e. not exponential in the characteristics of the problem such as the number of factors of variation or the number of inputs. These units are generally organized in layers so that the many levels of computation can be composed.

A function may appear complex from the point of view of a local non-parametric learning algorithm such as a Gaussian kernel machine, because it has many variations, such as the *sine* function. On the other hand, the Kolmogorov complexity of that function could be small, and it could be representable efficiently with a deep architecture. See Bengio and Le Cun (2007) for more discussion on this subject, and pointers to the circuit complexity theory literature showing that shallow circuits can require exponentially more components than deeper circuits.

However, optimizing deep architectures is computationally challenging. It was believed until recently impractical to train deep neural networks (except Convolutional Neural Networks (LeCun et al., 1989)), as iterative optimization procedures tended to get stuck near poor local minima. Fortunately, effective optimization procedures using unsupervised learning have recently been proposed and have demonstrated impressive performance for deep architectures.

### 1.2. Scaling to Harder Learning Problems

Though there are benchmarks to evaluate generic learning algorithms (e.g. the UCI Machine Learning Repository) many of these proposed learning problems do not possess the kind of complexity we address here.

We are interested in problems for which the underlying data distribution can be thought as the product of factor distributions, which means that a sample corresponds to a combination of particular values for these factors. For example, in a digit recognition task, the factors might be the scaling, rotation angle, deviation from the center of the image and the background of the image. Note how some of these factors (such as the background) may be very high-dimensional. In natural language processing, factors which influence the distribution over words in a document include topic, style and various characteristics of the author. In speech recognition, potential factors can be the gender of the speaker, the background noise and the amount of echo in the environment. In these important settings, it is not feasible to collect enough data to cover the input space effectively; especially when these factors vary independently.

Research in incorporating factors of variation into learning procedures has been abundant. A lot of the published results refer to learning invariance in the domain of digit recognition and most of these techniques are engineered for a specific set of invariances. For instance, Decoste and Scholkopf (2002) present a thorough review that discusses the problem of incorporating prior knowledge into the training procedure of kernel-based methods. More specifically, they discuss prior knowledge about invariances such as translations, rotations etc. Three main methods are described:

1. hand-engineered kernel functions,
2. artificial generation of transformed examples (the so-called *Virtual SV* method),
3. and a combination of the two: engineered kernels that generate artificial examples (e.g. *kernel jittering*).

The main drawback of these methods, from our point of view, is that domain experts are required to explicitly identify the types of invariances that need to be modeled. Furthermore these invariances are highly problem-specific. While there are cases for which manually crafted invariant features are readily available, it is difficult in general to construct invariant features.

We are interested in learning procedures and architectures that would *automatically* discover and represent such invariances (ideally, in an efficient manner). We believe that one good way of achieving such goals is to have procedures that learn high-level features ("abstractions") that build on lower-level features. One of the main goals of this paper is thus to examine empirically the link between high-level feature extraction and different types of invariances. We start by describing two architectures that are designed for extracting high-level features.

## 2. Learning Algorithms with Deep Architectures

Hinton et al. (2006) introduced a greedy layer-wise *unsupervised* learning algorithm for Deep Belief Networks (DBN). This training strategy for such networks was subsequently analyzed by Bengio et al. (2007) who concluded that it is an important ingredient in effective optimization and training of deep networks. While lower layers of a DBN extract "low-level features" from the input observation $\mathbf{x}$, the upper layers are supposed to represent more "abstract" concepts that explain $\mathbf{x}$.

### 2.1. Deep Belief Networks and Restricted Boltzmann Machines

For classification, a DBN model with $\ell$ layers models the joint distribution between target $y$, observed variables $x_j$ and $i$ hidden layers $\mathbf{h}^k$ made of all binary units $h_i^k$, as follows:

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

where $\mathbf{x} = \mathbf{h}^0$, $P(\mathbf{h}^k|\mathbf{h}^{k+1})$ has the form given by equation 1 and $P(y, \mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$ is a Restricted Boltzmann Machine (RBM), with the bottom layer being the concatenation of $y$ and $\mathbf{h}^{\ell-1}$ and the top layer is $\mathbf{h}^\ell$.

An RBM with $n$ hidden units is a parametric model of the joint distribution between hidden variables $h_i$ and observed variables $x_j$ of the form:

$$P(\mathbf{x}, \mathbf{h}) \propto e^{\mathbf{h}'W\mathbf{x} + b'\mathbf{x} + c'\mathbf{h}}$$

with parameters $\theta = (W, b, c)$. If we restrict $h_i$ and $x_j$ to be binary units, it is straightforward to show that

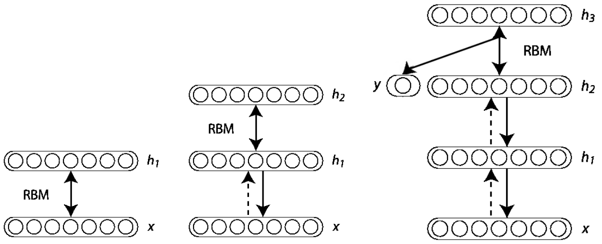
$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(x_i|\mathbf{h}) = \prod_i sigm(b_i + \sum_j W_{ji} h_j) \quad (1)$$

where *sigm* is the logistic sigmoid function, and $P(\mathbf{h}|\mathbf{x})$ also has a similar form:

$$P(\mathbf{h}|\mathbf{x}) = \prod_j P(h_j|\mathbf{x}) = \prod_j sigm(c_j + \sum_i W_{ji} x_i) \quad (2)$$

The RBM form can be generalized to other conditional distributions besides the binomial, including continuous variables. See Welling et al. (2005) for a generalization of RBM models to conditional distributions from the exponential family.

RBM models can be trained by gradient descent. Although $P(\mathbf{x})$ is not tractable in an RBM, the Contrastive Divergence gradient (Hinton, 2002) is a good stochastic approximation of $\frac{\partial \log P(\mathbf{x})}{\partial \theta}$. The contrastive divergence stochastic gradient can be used to initialize each layer of a DBN as an RBM. The number of layers can be increased greedily, with the newly added top layer trained as an RBM to model the output of the previous layers. When initializing the weights to $\mathbf{h}^\ell$, an RBM is trained to model the concatenation of $y$ and $\mathbf{h}^{\ell-1}$. This iterative pre-training procedure is illustrated in figure 2.



(a) RBM for $\mathbf{x}$ (b) RBM for $\mathbf{h}^1$ (c) RBM for $\mathbf{h}^2$ and $y$

*Figure 2.* Iterative pre-training construction of a Deep Belief Network.

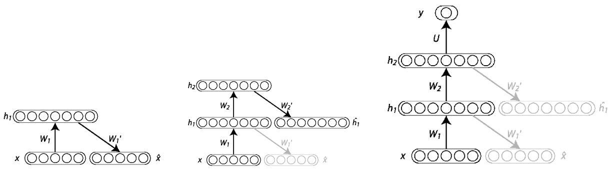
Using a mean-field approximation of the conditional distribution of layer $\mathbf{h}^{\ell-1}$, we can compute a representation $\widehat{\mathbf{h}}^{\ell-1}$ for the input by setting $\widehat{\mathbf{h}}^0 = \mathbf{x}$ and iteratively computing $\widehat{\mathbf{h}}^k = P(\mathbf{h}^k|\widehat{\mathbf{h}}^{k-1})$ using equation 2. We then compute the probability of all classes given the approximately inferred value $\widehat{\mathbf{h}}^{\ell-1}$ for $\mathbf{h}^{\ell-1}$ using the following expression:

$$P(y|\widehat{\mathbf{h}}^{\ell-1}) = \sum_{\mathbf{h}^\ell} P(y, \mathbf{h}^\ell | \widehat{\mathbf{h}}^{\ell-1})$$

which can be calculated efficiently. The network can then be fine-tuned according to this estimation of the class probabilities by maximizing the log-likelihood of the class assignments in a training set using standard back-propagation.

### 2.2. Stacked Autoassociators

As demonstrated by Bengio et al. (2007), the idea of successively extracting non-linear features that "explain" variations of the features at the previous level can be applied not only to RBMs but also to *autoassociators*. An autoassociator is simply a model (usually a one-hidden-layer neural network) trained to reproduce its input by forcing the computations to flow through a "bottleneck" representation. Here we used the following architecture for autoassociators. Let $\mathbf{x}$ be the input of the autoassociator, with $x_i \in [0, 1]$, interpreted as the probability for the bit to be 1. For a layer with weight matrix $W$, hidden biases column vector $b$ and input biases column vector $c$, the reconstruction probability for bit $i$ is $p_i(\mathbf{x})$, with the vector



(a) Reconst. $\mathbf{x}$ (b) Reconst. $\mathbf{h}^1$ (c) Predict $y$

*Figure 3.* Iterative training construction of the Stacked Autoassociators model.

of probabilities:

$$p(\mathbf{x}) = sigm(c + W sigm(b + W'\mathbf{x})).$$

The training criterion for the layer is the average of negative log-likelihoods for predicting $\mathbf{x}$ from $p(\mathbf{x})$. For example, if $\mathbf{x}$ is interpreted either as a sequence of bits or a sequence of bit probabilities, we minimize the reconstruction cross-entropy:

$$R = -\sum_i x_i \log p_i(\mathbf{x}) + (1 - x_i) \log(1 - p_i(\mathbf{x})).$$

See Bengio et al. (2007) for more details. Once an autoassociator is trained, its internal "bottleneck" representation (here, $sigm(b + W'\mathbf{x})$) can be used as the input for training a second autoassociator etc. Figure 3 illustrates this iterative training procedure. The stacked autoassociators can then be fine-tuned with respect to a supervised training criterion (adding a predictive output layer on top), using back-propagation to compute gradient on parameters of all layers.

## 3. Benchmark Tasks

In order to study the capacity of these algorithms to scale to learning problems with many factors of variation, we have generated datasets where we can identify some of these factors of variation explicitly. We focused on vision problems, mostly because they are easier to generate and analyze. In all cases, the classification problem has a balanced class distribution.

### 3.1. Variations on Digit Recognition

Models with deep architectures have been shown to perform competitively on the MNIST digit recognition dataset (Hinton et al., 2006; Bengio et al., 2007; Salakhutdinov & Hinton, 2007). In this series of experiments, we construct new datasets by adding additional factors of variation to the MNIST images. The generative process used to generate the datasets is as follows:

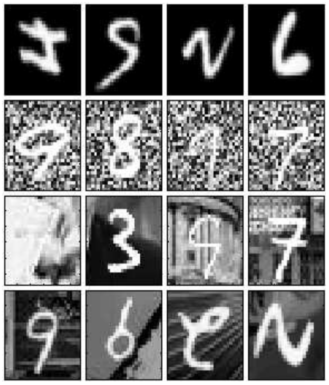


*Figure 4.* From top to bottom, samples from *mnist-rot*, *mnist-back-rand*, *mnist-back-image*, *mnist-rot-back-image*.

1. Pick sample $(x, y) \in \mathcal{X}$ from the digit recognition dataset;

2. Create a perturbed version $\widehat{x}$ of $x$ according to some factors of variation;

3. Add $(\widehat{x}, y)$ to a new dataset $\widehat{\mathcal{X}}$;

4. Go back to 1 until enough samples are generated.

Introducing multiple factors of variation leads to the following benchmarks:

*mnist-rot*: the digits were rotated by an angle generated uniformly between 0 and $2\pi$ radians. Thus the factors of variation are the rotation angle and those already contained in MNIST, such as hand writing style;

*mnist-back-rand*: a random background was inserted in the digit image. Each pixel value of the background was generated uniformly between 0 and 255;

*mnist-back-image*: a random patch from a black and white image was used as the background for the digit image. The patches were extracted randomly from a set of 20 images downloaded from the internet. Patches which had low pixel variance (i.e. contained little texture) were ignored;

*mnist-rot-back-image*: the perturbations used in *mnist-rot* and *mnist-back-image* were combined.

These 4 databases have 10000, 2000 and 50000 samples in their training, validation and test sets respectively. Figure 4 shows samples from these datasets.

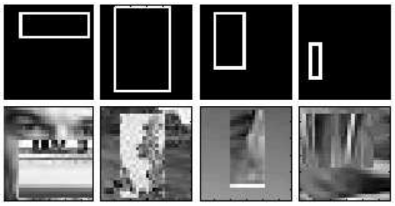


*Figure 5.* From top to bottom, samples from *rectangles* and *rectangles-image*.

### 3.2. Discrimination between Tall and Wide Rectangles

In this task, a learning algorithm needs to recognize whether a rectangle contained in an image has a larger width or length. The rectangle can be situated anywhere in the 28 × 28 pixel image. We generated two datasets for this problem:

*rectangles*: the pixels corresponding to the border of the rectangle has a value of 255, 0 otherwise. The height and width of the rectangles were sampled uniformly, but when their difference was smaller than 3 pixels the samples were rejected. The top left corner of the rectangles was also sampled uniformly, constrained so that the whole rectangle would fit in the image;

*rectangles-image*: the border and inside of the rectangles corresponds to an image patch and a background patch is also sampled. The image patches are extracted from one of the 20 images used for *mnist-back-image*. Sampling of the rectangles is essentially the same as for *rectangles*, but the area covered by the rectangles was constrained to be between 25% and 75% of the total image, the length and width of the rectangles were forced to be of at least 10 and their difference was forced to be of at least 5 pixels.

We generated training sets of size 1000 and 10000 and validation sets of size 200 and 2000 for *rectangles* and *rectangles-image* respectively. The test sets were of size 50000 in both cases. Samples for these two tasks are displayed in figure 5.

### 3.3. Recognition of Convex Sets

The task of discriminating between tall and wide rectangles was designed to exhibit the learning algorithms' ability to process certain image shapes and learn their properties. Following the same principle, we designed another learning problem which consists in indicating if a set of pixels forms a convex set.

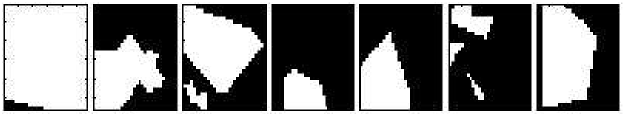


*Figure 6.* Samples from *convex*, where the first, fourth, fifth and last samples correspond to convex white pixel sets.

Like the MNIST dataset, the convex and non-convex datasets both consist of images of $28 \times 28$ pixels. The convex sets consist of a single convex region with pixels of value 255 (white). Candidate convex images were constructed by taking the intersection of a random number of half-planes whose location and orientation were chosen uniformly at random.

Candidate non-convex images were constructed by taking the union of a random number of convex sets generated as above. The candidate non-convex images were then tested by checking a convexity condition for every pair of pixels in the non-convex set. Those sets that failed the convexity test were added to the dataset. The parameters for generating the convex and non-convex sets were balanced to ensure that the mean number of pixels in the set is the same.

The generated training, validation and test sets are of size 6000, 2000 and 50000 respectively. Samples for this tasks are displayed in figure 6.

## 4. Experiments

We performed experiments on the proposed benchmarks in order to compare the performance of models with deep architectures with other popular generic classification algorithms.

In addition to the Deep Belief Network (denoted DBN-3) and Stacked Autoassociators (denoted SAA-3) models, we conducted experiments with a single hidden-layer DBN (DBN-1), a single hidden-layer neural network (NNet), SVM models with Gaussian ($\mathbf{SVM}_{rbf}$) and polynomial ($\mathbf{SVM}_{poly}$) kernels.

In all cases, model selection was performed using a validation set. For NNet, the best combination of number of hidden units (varying from 25 to 700), learning rate (from 0.0001 to 0.1) and decrease constant (from 0 to $10^{-6}$) of stochastic gradient descent and weight decay penalization (from 0 to $10^{-5}$) was selected using a grid search.

For DBN-3 and SAA-3, both because of the large number of hyper-parameters and because these models can necessitate more than a day to train, we could not perform a full grid search in the space of hyper-parameters. For both models, the number of hidden units per layer must be chosen, in addition to all other optimization parameters (learning rates for the unsupervised and supervised phases, stopping criteria of the unsupervised phase, etc.). The hyper-parameter search procedure we used alternates between fixing a neural network architecture and searching for good optimization hyper-parameters in a manner similar to coordinate descent. See `http://www.iro.umontreal.ca/~lisa/icml2007` for more details about this procedure. In general, we tested from 50 to 150 different configurations of hyper-parameters for DBN-3 and SAA-3. The layer sizes varied in the intervals $[500, 3000]$, $[500, 4000]$ and $[1000, 6000]$ respectively for the first, second and third layer and the learning rates varied between 0.0001 and 0.1. In the case of the single hidden layer DBN-1 model, we allowed ourselves to test for much larger hidden layer sizes, in order to balance the number of parameters between it and the DBN-3 models we tested.

For all neural networks, we used early stopping based on the classification error of the model on the validation set. However during the initial unsupervised training of DBN-3, the intractability of the RBM training criterion precluded the use of early stopping. Instead, we tested 50 or 100 unsupervised learning epochs for each layer and selected the best choice based on the final accuracy of the model on the validation set.

The experiments with the NNet, DBN-1, DBN-3 and SAA-3 models were conducted using the `PLearn`[1] library, an Open Source C++ library for machine learning which was developed and is actively used in our lab.

In the case of SVMs with Gaussian kernels, we performed a two-stage grid search for the width of the kernel and the soft-margin parameter. In the first stage, we searched through a coarse logarithmic grid ranging from $\sigma = 10^{-7}$ to 1 and $C = 0.1$ to $10^5$. In the second stage, we performed a more fine-grained search in the vicinity of that tuple $(\sigma, C)$ that gave the best validation error. In the case of the polynomial kernel, the strategy was the same, except that we searched through all possible degrees of the polynomial up to 20, rendering the fine-grained search on this parameter useless. Conforming to common practice, we also allowed the SVM models to be retrained on the concatenation of the training and validation set using the selected hyper-parameters. Throughout the experiments we used the publicly available library `libSVM` (Chang & Lin, 2001), version 2.83.

For all datasets, the input was normalized to have values between 0 and 1. When the input was binary (i.e. for *rectangles* and *convex*), the Deep Belief Network model used binary input units and when the input was in $[0, 1]^n$ (i.e. for *mnist-rot*, *mnist-back-rand*, *mnist-back-imag*, *mnist-rot-back-image* and *rectangles-image*) it used truncated exponential input units (Bengio et al., 2007).

### 4.1. Benchmark Results

The classification performances for the different learning algorithms on the different datasets of the benchmark are reported in table 1. As a reference for the variations on digit recognition experiments, we also include the algorithms' performance on the original MNIST database, with training, validation and test sets of size 10000, 2000 and 50000 respectively. Note that the training set size is significantly smaller than that typically used.

There are several conclusions which can be drawn from these results. First, taken together, deep architecture models show globally the best performance. Seven times out of 8, either DBN-3 or SAA-3 are among the best performing models (within the confidence intervals). Four times out of 8 the best accuracy is obtained with a deep architecture model (either DBN-3 or SAA-3). This is especially true in three cases: *mnist-back-rand*, *mnist-back-image* and *mnist-rot-back-image*, where they perform better by a large margin. Also, deep architecture models consistently improve on NNet, which is basically a shallow and totally supervised version of the deep architecture models.

Second, the improvement provided by deep architecture models is most notable for factors of variation related to background, especially in the case of random background, where DBN-3 almost reaches its performance on *mnist-basic*. It seems however that not all of the invariances can be learned just as easily–an example is the one of rotation, where the deep architectures do not outperform SVMs. $\mathbf{SVM}_{rbf}$ does achieve an impressive result; we believe that this is possible because of the large number of samples in the training set (the input space is well populated) and because there is only one factor applied (contrast this with the score we obtain with $\mathbf{SVM}_{rbf}$ on *mnist-rot-back-image* where the presence of two factors creates a less well-behaved input space)

Third, even though SAA-3 and DBN-3 provide con-

[1] See `http://www.plearn.org/`

*Table 1.* Results on the benchmark for problems with factors of variation (in percentages). The best performance as well as those with overlapping confidence intervals are marked in bold.

| **Dataset** | **SVM$_{rbf}$** | **SVM$_{poly}$** | **NNet** | **DBN-1** | **SAA-3** | **DBN-3** |
|---|---|---|---|---|---|---|
| *mnist-basic* | **3.03±0.15** | 3.69±0.17 | 4.69±0.19 | 3.94±0.17 | 3.46±0.16 | **3.11±0.15** |
| *mnist-rot* | **10.38±0.27** | 13.61±0.30 | 17.62±0.33 | 12.11±0.29 | 11.43±0.28 | 12.30±0.29 |
| *mnist-back-rand* | 14.58±0.31 | 16.62±0.33 | 20.04±0.35 | 9.80±0.26 | 11.28±0.28 | **6.73±0.22** |
| *mnist-back-image* | 22.61±0.37 | 24.01±0.37 | 27.41±0.39 | **16.15±0.32** | 23.00±0.37 | **16.31±0.32** |
| *mnist-rot-back-image* | 32.62±0.41 | 37.59±0.42 | 42.17±0.43 | 31.84±0.41 | **24.09±0.37** | 28.51±0.40 |
| *rectangles* | **2.15±0.13** | **2.15±0.13** | 7.16±0.23 | 4.71±0.19 | **2.41±0.13** | 2.60±0.14 |
| *rectangles-image* | 24.04±0.37 | 24.05±0.37 | 33.20±0.41 | 23.69±0.37 | 24.05±0.37 | **22.50±0.37** |
| *convex* | 19.13±0.34 | 19.82±0.35 | 32.25±0.41 | 19.92±0.35 | **18.41±0.34** | **18.63±0.34** |

sistent improvement over NNet, these models are still sensitive to hyper-parameter selection. This might explain the surprising similarity of the results for SAA-3 on *mnist-back-image* and *mnist-rot-back-image*, even though the former corresponds to an easier learning problem than the latter.

## 4.2. Impact of Background Pixel Correlation

Looking at the results obtained on *mnist-back-rand* and *mnist-back-image* by the different algorithms, it seems that pixel correlation contained in the background images is the key element that worsens the performances. To explore the disparity in performance of the learning algorithms between MNIST with independent noise and MNIST on a background image datasets, we made a series of datasets of MNIST digits superimposed on a background of correlated noisy pixel values.

Correlated pixel noise was sampled from a zero-mean multivariate Gaussian distribution of dimension equal to the number of pixels: $s \sim \mathcal{N}(0, \Sigma)$. The covariance matrix, $\Sigma$, is specified by a convex combination of an identity matrix and a Gaussian kernel function (with bandwidth $\sigma = 6$) with mixing coefficient $\gamma$. The Gaussian kernel induced a neighborhood correlation structure among pixels such that nearby pixels are more correlated than pixels further apart. For each sample from $\mathcal{N}(0, \Sigma)$, the pixel values $p$ (ranging from 0 to 255) were determined by passing elements of $s$ through the standard error function $p_i = erf(s_i/\sqrt{2})$ and multiplying by 255. We generated six datasets with varying degrees of neighborhood correlation by setting the mixture weight $\gamma$ to the values $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The marginal distributions for each pixel $p_i$ is uniform[0,1] for each value of $\gamma$. Figure 7 shows some samples from the 6 different tasks.

We ran experiments on these 6 datasets, in order to measure the impact of background pixel correlation on the classification performance. Figure 8 shows a comparison of the results obtained by DBN-3, SAA-3 and $SVM_{rbf}$. In the case of the deep models, we used the same layer sizes for all six experiments. The selected layer sizes had good performance on both *mnist-back-image* and *mnist-back-rand*. However, we did vary the hyper-parameters related to the optimization of the deep networks and chose the best ones for each problems based on the validation set performance. All hyper-parameters of $SVM_{rbf}$ were chosen according to the same procedure.

It can be seen that, as the amount of background pixel correlation increases, the classification performance of

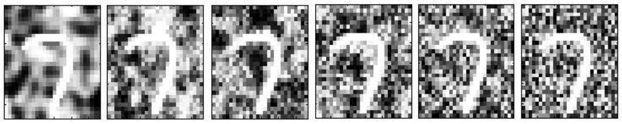


*Figure 7.* From left to right, samples with progressively less pixel correlation in the background.

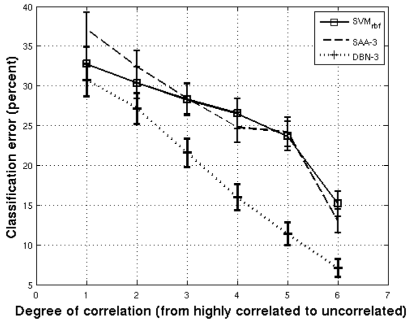


*Figure 8.* Classification error of **SVM$_{rbf}$**, **SAA-3** and **DBN-3** on MNIST examples with progressively less pixel correlation in the background.

all three algorithms degrade. This is coherent with the results obtained on *mnist-back-image* and *mnist-back-rand*. This also indicates that, as the factors of variation become more complex in their interaction with the input space, the relative advantage brought by DBN-3 and SAA-3 diminishes. This observation is preoccupying and implies that learning algorithms such as DBN-3 and SAA-3 will eventually need to be adapted in order to scale to harder, potentially "real life" problem.

One might argue that it is unfair to maintain the same layer sizes of the deep architecture models in the previous experiment, as it is likely that the model will need more capacity as the input distribution becomes more complex. This is a valid point, but given that, in the case of DBN-3 we already used a fairly large network (the first, second and third layers had respectively 3000, 2000 and 2000 hidden units), scaling the size of the network to even bigger hidden layers implies serious computational issues. Also, for even more complex datasets such as the NORB dataset (LeCun et al., 2004), which consists in 108 × 108 stereo images of objects from different categories with many factors of variation such as lighting conditions, elevation, azimuth and background, the size of the deep models becomes too large to even fit in memory. In our preliminary experiments where we subsampled the images to be $54 \times 54$ pixels, the biggest models we were able to train only reached 51.6% (DBN-3) and 48.0% (SAA-3), whereas $SVM_{rbf}$ reached 43.6% and NNet reached 43.2%. Hence, a natural next step for learning algorithms for deep architecture models would be to find a way for them to use their capacity to more directly model features of the data that are more predictive of the target value.

Further details of our experiments and links to downloadable versions of the datasets are available online at: `http://www.iro.umontreal.ca/~lisa/icml2007`

## 5. Conclusion and Future Work

We presented a series of experiments which show that deep architecture models tend to outperform other shallow models such as SVMs and single hidden-layer feed-forward neural networks. We also analyzed the relationships between the performance of these learning algorithms and certain properties of the problems that we considered. In particular, we provided empirical evidence that they compare favorably to other state-of-the-art learning algorithms on learning problems with many factors of variation, but only up to a certain point where the data distribution becomes too complex and computational constraints become an important issue.

## Acknowledgments

We would like to thank Yann LeCun for suggestions and discussions. We thank the anonymous reviewers who gave useful comments that improved the paper. This work was supported by NSERC, MITACS and the Canada Research Chairs.

## References

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems 19*. MIT Press.

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

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Decoste, D., & Scholkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*, 161–190.

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

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

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

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

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

LeCun, Y., Huang, F.-J., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of CVPR'04*. IEEE Press.

Salakhutdinov, R., & Hinton, G. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. *To Appear in Proceedings of AISTATS'2007*.

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