Why does Unsupervised Pre-training Help Deep Learning?

Dumitru Erhan[*] **dumitru.erhan@umontreal.ca**
Yoshua Bengio **yoshua.bengio@umontreal.ca**
Aaron Courville **aaron.courville@umontreal.ca**
Pierre-Antoine Manzagol **pierre-antoine.manzagol@umontreal.ca**
Pascal Vincent **pascal.vincent@umontreal.ca**
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour, Montréal, Québec, H3T 1J8, Canada

Samy Bengio **bengio@google.com**
Google Research
1600 Amphitheatre Parkway, Mountain View, CA, 94043, USA

Editor:

Abstract

Much recent research has been devoted to learning algorithms for deep architectures such as Deep Belief Networks and stacks of auto-encoder variants, with impressive results obtained in several areas, mostly on vision and language datasets. The best results obtained on supervised learning tasks involve an unsupervised learning component, usually in an unsupervised pre-training phase. Even though these new algorithms have enabled training deep models, many questions remain as to the nature of this difficult learning problem. The main question investigated here is the following: why does unsupervised pre-training work and why does it work so well? Answering these questions is important if learning in deep architectures is to be further improved. We propose several explanatory hypotheses and test them through extensive simulations. We empirically show the influence of pre-training with respect to architecture depth, model capacity, and number of training examples. The experiments confirm and clarify the advantage of unsupervised pre-training. The results suggest that unsupervised pre-training guides the learning towards basins of attraction of minima that are better in terms of the underlying data distribution; the evidence from these results supports a regularization explanation for the effect of pre-training.

Keywords: Deep architectures, unsupervised pre-training, deep belief networks, stacked denoising auto-encoders, neural networks, non-convex optimization.

1. Introduction

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Theoretical results (Yao, 1985; Håstad, 1986; Håstad and Goldmann, 1991; Bengio et al., 2006), reviewed

[*]. Part of this work was done while Dumitru Erhan was at Google Research

and discussed by Bengio and LeCun (2007), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*. The recent surge in experimental work in the field seems to support this notion, accumulating evidence that in challenging AI-related tasks – such as computer vision (Bengio et al., 2007; Ranzato et al., 2007; Larochelle et al., 2007; Ranzato et al., 2008; Lee et al., 2009; Mobahi et al., 2009; Osindero and Hinton, 2008), natural language processing (NLP) (Collobert and Weston, 2008; Weston et al., 2008), robotics (Hadsell et al., 2008), or information retrieval (Salakhutdinov and Hinton, 2007; Salakhutdinov et al., 2007) – deep learning methods significantly out-perform comparable but shallow competitors, and often match or beat the state-of-the-art.

These recent demonstrations of the potential of deep learning algorithms were achieved despite the serious challenge of training models with many layers of adaptive parameters. In virtually all instances of deep learning, the objective function is a highly non-convex function of the parameters, with the potential for many distinct *local minima* in the model parameter space. The principal difficulty is that not all of these minima provide equivalent generalization errors and, we suggest, that for deep architectures, the standard training schemes tend to place the parameters in regions of the parameters space that generalize poorly – as was frequently observed empirically but rarely reported (Bengio and LeCun, 2007).

The breakthrough to effective training strategies for deep architectures came in 2006 with the algorithms for training deep belief networks (DBN) (Hinton et al., 2006) and stacked auto-encoders (Ranzato et al., 2007; Bengio et al., 2007), which are all based on a similar approach: greedy layer-wise unsupervised pre-training followed by supervised fine-tuning. Each layer is pre-trained with an unsupervised learning algorithm, learning a nonlinear transformation of its input (the output of the previous layer) that captures the main variations in its input. This unsupervised pre-training sets the stage for a final training phase where the deep architecture is fine-tuned with respect to a supervised training criterion with a gradient-based optimization. While the improvement in performance of trained deep models offered by the pre-training strategy is impressive, little is understood about the mechanisms underlying this success.

The objective of this paper is to explore, through extensive experimentation, how unsupervised pre-training works to render learning deep architectures more effective and why they appear to work so much better than traditional neural network training methods. There are a few reasonable hypotheses why pre-training might work. One possibility that pre-training acts as a kind of network pre-conditioner, putting the parameter values in the appropriate range for further supervised training. Another possibility, suggested by Bengio et al. (2007), is that pre-training initializes the model to a point in parameter space that somehow renders the optimization process more effective, in the sense of achieving a lower minimum of the empirical cost function.

Here, we argue that our experiments support a view of pre-training as an unusual form of *regularization*: minimizing variance and introducing bias towards configurations of the parameter space that are useful for unsupervised learning. This perspective places pre-training well within the family of recently developed semi-supervised methods. The pre-training approach is, however, unique among semi-supervised training strategies in that it acts by defining a particular initialization point for standard supervised training rather than either modifying the supervised objective function (Barron, 1991) or explicitly imposing constraints on the parameters throughout training (Lasserre et al., 2006). This type of initialization-as-regularization strategy has precedence in the neural networks literature, in the shape of the early stopping idea (Sjöberg and Ljung, 1995; Amari et al., 1997). We suggest that, in the highly non-convex situation of training a deep architecture, defining a particular initialization point *implicitly* imposes constraints on the parameters in that it specifies which minimum (of a very large number of possible minima) of the cost function is sought. In this way, it may be possible to think of the pre-training as being related to the approach of Lasserre et al. (2006).

Another important and distinct property of the pre-training strategy is that in the standard situation of training using stochastic gradient descent, the beneficial generalization effects due to pre-training do not appear to diminish as the number of labeled examples grows very large. We argue that this is a consequence of the combination of the non-convexity (multi-modality) of the objective function and the dependency of the stochastic gradient descent method on example ordering. We found that early changes in the parameters (when they are still small) have a greater impact on the final region (basin of attraction of the descent procedure) in which the learner ends up. In particular, unsupervised pre-training sets the parameter in a region from which better basins of attraction can be reached, both from the standpoint of the training criterion (especially for large datasets) and in terms of generalization. Hence, although pre-training is a regularizer, it can have a positive effect on the training objective when the number of training examples is large.

As previously stated, this paper is concerned with an experimental assessment of the various competing hypotheses regarding the role of pre-training in the recent success of deep learning methods. To this end, we present a series of experiments design to pit these hypotheses (mostly the regularization hypothesis and the optimization hypothesis stated in previous papers, along with the hypothesis summarized above) against one another in an attempt to resolve at least some of the mystery surrounding the effectiveness of unsupervised pre-training.

In the first set of experiments (in Section 6), we establish the effect of pre-training on improving the generalization error of trained deep architectures. In this section we also exploit dimensionality reduction techniques to illustrate how pre-training affects the location of minima in parameter space. In the second set of experiments (in Section 7), we directly compare the two alternative hypotheses (pre-trainind as a pre-conditioner; and pre-training as an optimization scheme) against the hypothesis that pre-training is a regularization strategy. In the final set of experiments, (in Section 8), we explore the role of pre-training in the online learning setting, where the number of available training examples grows very large. In these experiments, we test key aspects of our hypothesis relating to the topology of the cost function and the role of pre-training in manipulating the region of parameter space from which supervised training is initiated. However, before delving into the experiments, we begin with a more in-depth view of the challenges in training deep architectures and how we believe pre-training works to overcome these challenges.

2. The Challenges of Deep Learning

In this section, we present a perspective on why standard training of deep models through gradient backpropagation appears to be so difficult. First, it is important to establish what we mean in stating that training is difficult. What complicates matters is that different levels in the hierarchy may be more or less difficult to train. For example the training criterion is generally convex in the parameters of the top layer, and training one-hidden layer neural networks is generally very successfull (in spite of the potential for local minima), hence the higher layers (near the output) are easy to optimize, given the lower layers. However, given the higher layers as fixed, we conjecture that training the lower layers is a difficult optimization problem, and trying to optimize both simultaneously may be even more difficult. Furthermore, because with enough capacity the top two layers can easily overfit the training set, training error does not necessarily reveal the difficulty in optimization the lower layers. This is where unsupervised pre-training seems to make a big difference, by setting parameters of the lower layers in a region from which good solutions (in terms of generalization error) can be found.

We believe the central challenge in training deep architectures is dealing with the strong dependencies that exist during training between the parameters across layers. One way to conceive the difficulty of the problem is that we must simultaneously:

1. adapt the lower layers in order to provide adequate input to the final (end of training) setting of the upper layers

2. adapt the upper layers to make good use of the final (end of training) setting of the lower layers.

The second problem is easy on its own (ie, when the final setting of the other layers is known). It is not clear how difficult is the first one, and we conjecture that a particular difficulty arises when both sets of layers must be learned jointly, as the gradient of the objective function is limited to a local measure given the current setting of other parameters. As shown in our experiments here, the standard training schemes tend to place the parameters in regions of the parameters space that generalize poorly.

A separate but related issue appears if we focus our consideration of traditional training methods for deep architectures on stochastic gradient descent. A sequence of examples along with an online gradient descent procedure defines a trajectory in parameter space, which converges in some sense (the error does not improve anymore, maybe because we are near a local minimum). The hypothesis is that small perturbations of that trajectory (either by initialization or by changes in which examples are seen when) have more effect early on. Early in the process of following the stochastic gradient, changes in the weights tend to increase their magnitude and the amount of non-linearity of the network. As this happens, the set of regions accessible by stochastic gradient descent on samples of the training distribution becomes smaller. This is consistent with the idea that there are basins of attraction of the dynamics of learning, and that early on small perturbations allow to switch from one basin to a nearby one, whereas later on (typically with larger parameter values), it is unlikely to "escape" from such a basin of attraction. Hence the early examples can have a larger influence and, in practice, trap the model parameters in particular regions of parameter space that correspond to the specific and arbitrary ordering of the training examples[1]. An important consequence of this phenomenon, is that even in the presence of a very large (effectively infinite) amount of supervised data, stochastic gradient descent is subject to a degree of *overfitting* to the training data presented early in the training process. In that sense, pre-training is a regularizer which interacts intimately with the optimization process, and when the number of training examples becomes large, its positive effect is seen not only on generalization error but also on training error.

3. Pre-training Acts as a Regularizer

As stated in the introduction, we believe that greedy layer-wise pre-training overcomes the challenges of deep learning by introducing a useful prior to the supervised *fine-tuning* training procedure. We claim that the regularization effect is a consequence of the pre-training procedure establishing an initialization point of the fine-tuning procedure inside a region of parameter space in which the parameters are henceforth restricted. The parameters are restricted to a relatively small volume of parameter space that is delineated by the boundary of the *local basin of attraction* of the supervised fine-tuning cost function.

The pre-training procedure increases the magnitude of the weights and in standard deep models, with a sigmoidal nonlinearity, this has the effect of rendering both the function more nonlinear and the cost function locally more complicated with more topological features such as peaks, troughs and plateaus. The existence of these topological features renders the parameter space locally more difficult to travel significant distances via a gradient descent procedure. This is the core of the restrictive property imposed by the pre-training procedure and hence the basis of its regularizing properties.

But unsupervised pre-training does not restrict to any such region: it restricts in a way that can bring useful information from the input distribution $P(X)$. To simply state that pre-training is a regularization strategy somewhat undermines the significance of its effectiveness. Not all regularizers are created equal and, in comparison to standard regularization schemes such as L_1 and L_2 parameter penalization, pre-training is dramatically effective. We believe the credit for its success can be attributed to the unsupervised training criteria optimized during pre-training.

During each phase of the greedy unsupervised training strategy, the currently trained layer learns to represent, by way of one of a variety of cost functions (that usually correspond roughly to a reconstruction error), the dominant factors of variation extant in the data. This has the effect of leveraging knowledge of X to form, at each layer, a representation of X that consists of statistically reliable features of X that can then be used to predict the output (usually a class label) Y. This perspective places pre-training well within the family of learning strategies collectively know as semi-supervised methods. As with other recent work demonstrating the effectiveness of semi-supervised methods in regularizing model parameters, we claim that the effectiveness of the pre-training strategy is limited to extent that learning $P(X)$ is helpful in learning $P(Y|X)$. Here, we find transformations of X that are predictive of the main factors of variation in $P(X)$, and when the prior is effective these transformations of X are also predictive of Y. In the context of deep learning, the greedy unsupervised strategy may also have a special function. To

1. This process seems similar to the "critical period" phenomena observed in neuroscience and psychology (Bornstein, 1987)

some degree it resolves the problem of simultaneously learning the parameters at all layers (mentioned in Section 2) by introducing a proxy criterion that allows the significant factors of variation, present in the data, to be communicated throughout the model. As a result, during early stages of learning, the upper layers (those that typically learn quickly) have access to a more robust representation of the input and are less likely to key in on spurious aspect of the data in fitting the target function.

To clarify this line of reasoning, we can formalize the effect of pre-training in inducing a prior distribution over the parameters. Let us assume that parameters are forced to be chosen in a bounded region $\mathcal{S} \subset \mathbb{R}^d$. Let $\mathcal{S}$ be split in regions $\{R_k\}$ that are the basins of attraction of descent procedures in the training error (note that $\{R_k\}$ depends on the training set, but the dependency decreases as the number of examples increases). We have $\cup_k R_k = \mathcal{S}$ and $R_i \cap R_j = \emptyset$ for $i \neq j$. Let $v_k = \int 1_{\theta \in R_k} d\theta$ be the volume associated with region R_k (where θ are our model's parameters). Let r_k be the probability that a purely random initialization (according to our initialization procedure, which factorizes across parameters) lands in R_k, and let π_k be the probability that pre-training (following a random initialization) lands in R_k, i.e. $\sum_k r_k = \sum_k \pi_k = 1$. We can now take into account the initialization procedure as a regularization term:

$$\text{regularizer} = -\log P(\theta). \tag{1}$$

For pre-trained models, the prior is

$$P_{\text{pre-training}}(\theta) = \sum_k 1_{\theta \in R_k} \pi_k / v_k. \tag{2}$$

For the models without pre-training, the prior is

$$P_{\text{no-pre-training}}(\theta) = \sum_k 1_{\theta \in R_k} r_k / v_k. \tag{3}$$

One can verify that $P_{\text{pre-training}}(\theta \in R_k) = \pi_k$ and $P_{\text{no-pre-training}}(\theta \in R_k) = r_k$. When π_k is tiny, the penalty is high when $\theta \in R_k$, with pre-training. The derivative of this regularizer is zero almost everywhere because we have chosen a uniform prior inside each region R_k. Hence, to take the regularizer into account, and having a generative model for $P_{\text{pre-training}}(\theta)$ (the pre-training procedure), it is reasonable to sample an initial θ from it (knowing that from this point on the penalty will not increase during the iterative minimization of the training criterion), and this is exactly how the pre-trained models are obtained in our experiments.

4. Previous Relevant Work

We start with an overview of the literature on semi-supervised learning (SSL), since the SSL framework is essentially the one in which we operate as well.

4.1 Related Semi-Supervised Methods

It has been recognized for some time that generative models are less prone to overfitting than discriminant ones (Ng and Jordan, 2001). Consider input variable X and target variable Y.

Whereas a discriminant model focuses on $P(Y|X)$, a generative model focuses on $P(X,Y)$ (often parametrized as $P(X|Y)P(Y)$), i.e., it also cares about getting $P(X)$ right, which can reduce the freedom of fitting the data when the ultimate goal is only to predict Y given X.

Exploiting information about $P(X)$ to improve generalization of a classifier has been the driving idea behind semi-supervised learning (Chapelle et al., 2006). For example, one can use unsupervised learning to map X into a representation (also called embedding) such that two examples $\mathbf{x}_1$ and $\mathbf{x}_2$ that belong to the same cluster (or are reachable through a short path going through neighboring examples in the training set) end up having nearby embeddings. One can then use supervised learning (e.g. a linear classifier) in that new space and achieve better generalization in many cases (Belkin and Niyogi, 2002; Chapelle et al., 2003). A long-standing variant of this approach is the application of Principal Components Analysis as a preprocessing step before applying a classifier (on the projected data). In these models the data is first transformed in a new representation using unsupervised learning, and a supervised classifier is stacked on top, learning to map the data in this new representations into class predictions.

Instead of having separate unsupervised and supervised components in the model, one can consider models in which $P(X)$ (or $P(X,Y)$) and $P(Y|X)$ share parameters (or whose parameters are connected in some way), and one can trade-off the supervised criterion $-\log P(Y|X)$ with the unsupervised or generative one ($-\log P(X)$ or $-\log P(X,Y)$). It can then be seen that the generative criterion corresponds to a particular form of prior (Lasserre et al., 2006), namely that the structure of $P(X)$ is connected to the structure of $P(Y|X)$ in a way that is captured by the shared parametrization. By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or a purely discriminative training criterion (Lasserre et al., 2006; Larochelle and Bengio, 2008).

In the context of deep architectures, a very interesting application of these ideas involves combining an unsupervised embedding criterion at each layer (or only one intermediate layer) with a traditional supervised criterion (Weston et al., 2008). This has been shown to be a powerful semi-supervised learning strategy, and is an alternative to the kind of algorithms described and evaluated in this paper, which also combine unsupervised learning with supervised learning.

4.2 Early Stopping as a Form of Regularization

We stated that pre-training as initialization can be seen as restricting the optimization procedure to a relatively small volume of parameter space that corresponds to a local basin of attraction of the supervised cost function. Early stopping can be seen as having a similar effect, by constraining the optimization procedure to a region of the parameter space that is close to the initial configuration of parameters.

Early stopping is considered a form of regularization. With τ the number of training iterations and η the learning rate used in the update procedure, $\tau\eta$ can be seen as the reciprocal of a regularization parameter. Indeed, restricting either quantity restricts the area of parameter space reachable from the starting point. In the case of the optimization of a simple linear model (initialized at the origin) using a quadratic error function and simple gradient descent, early stopping will have a similar effect to traditional

Thus, in both pre-training and early stopping, the parameters[2] are constrained (either implicitly or explicitly) to be close their initial values and both can then be seen as regularizers. A more formal treatment of early stopping as regularization is given by Sjöberg and Ljung (1995) and Amari et al. (1997). There is no equivalent treatment of pre-training, but this paper sheds some light on the effects of such initialization in the case of deep architectures.

5. Experimental Setup and Methodology

In this section, we describe the setting in which we test the hypothesis introduced in Section 3 and previously proposed hypotheses. This includes a description of the deep architectures used, the datasets and the details necessary to reproduce our results.

5.1 Models

All of the successful methods (Hinton et al., 2006; Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Vincent et al., 2008; Weston et al., 2008; Lee et al., 2008) in the literature for training deep architectures have something in common: they rely on an unsupervised learning algorithm that provides a training signal at the level of a single layer. Most work in two main phases. In a first phase, *unsupervised pre-training*, all layers are initialized using this layer-wise unsupervised learning signal. In a second phase, *fine-tuning*, a global training criterion (a prediction error, using labels in the case of a supervised task) is minimized. In the algorithms initially proposed (Hinton et al., 2006; Bengio et al., 2007), the unsupervised pre-training is done in a greedy layer-wise fashion: at stage k, the k-th layer is trained (with respect to an unsupervised criterion) using as input the output of the previous layer, and while the previous layers are kept fixed.

We shall consider two deep architectures as representatives of two families of models encountered in the deep learning literature.

5.1.1 Deep Belief Networks

The first model is the Deep Belief Net (DBN) by Hinton et al. (2006), obtained by training and stacking several layers of Restricted Boltzmann Machines (RBM) in a greedy manner. Once this stack of RBMs is trained, it can be used to initialize a multi-layer neural network for classification.

An RBM with n hidden units is a Markov Random Field (MRF) for the joint distribution between hidden variables h_i and observed variables x_j such that $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ factorize, i.e., $P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h}) = \prod_j P(x_j|\mathbf{h})$. The sufficient statistics of the MRF are typically h_i, x_j and $h_i x_j$, which gives rise to the following joint distribution:

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

with corresponding parameters $\theta = (W, b, c)$ (with $'$ denoting transpose, c_i associated with h_i, b_j with x_j, and W_{ij} with $h_i x_j$). If we restrict h_i and x_j to be binary units, it is

2. Of the supervised cost function.

straightforward to show that

$$\begin{aligned} P(\mathbf{x}|\mathbf{h}) &= \prod_j P(x_j|\mathbf{h}) \quad \text{with} \\ P(x_j = 1|\mathbf{h}) &= \text{sigmoid}(b_j + \sum_i W_{ij} h_i). \end{aligned} \tag{4}$$

where $\text{sigmoid}(\mathbf{a}) = 1/(1+\exp(-\mathbf{a}))$ (applied element-wise on a vector $\mathbf{a}$), and $P(\mathbf{h}|\mathbf{x})$ also has a similar form:

$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x}) \quad \text{with}$$

$$P(h_i|\mathbf{x}) = \text{sigmoid}(c_i + \sum_j W_{ij} x_j). \tag{5}$$

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

RBM models can be trained by approximate stochastic gradient descent. Although $P(\mathbf{x})$ is not tractable in an RBM, the Contrastive Divergence estimator (Hinton, 2002) is a good stochastic approximation of $\frac{\partial \log P(\mathbf{x})}{\partial \theta}$, in that it very often has the same sign (Bengio and Delalleau, 2009).

A DBN is a multi-layer generative model with layer variables h_0 (the input or visible layer), h_1, h_2, etc. The top two layers have a joint distribution which is an RBM, and $P(h_k|h_{k+1})$ are parametrized in the same way as for an RBM. Hence a 2-layer DBN is a DBN, and a stack of RBMs share parametrization with a corresponding DBN. The contrastive divergence update direction can be used to initialize each layer of a DBN as an RBM. Consider the first layer of the DBN trained as an RBM P_1 with hidden layer h_1 and visible layer v_1. We can train a second RBM P_2 that models (in its visible layer) the samples h_1 from $P_1(h_1|v_1)$ when v_1 is sampled from the training data set. It can be shown that this maximizes a lower bound on the log-likelihood of the DBN. The number of layers can be increased greedily, with the newly added top layer trained as an RBM to model the samples produced by chaining the posteriors $P(h_k|h_{k-1})$ of the lower layers (starting from h_0 from the training data set).

The parameters of DBN or of a stack of RBMs correspond to the parameters of a deterministic feedforward multi-layer neural network. The i-th unit of the k-th layer of the neural network outputs $\hat{h}_{ki} = \text{sigmoid}(c_{ki} + \sum_j W_{kij} \hat{h}_{k-1,j})$, using the parameters c_k and W_k of the k-th layer of the DBN. Hence, once the stack of RBMs or the DBN is trained, one can use those parameters to initialize the first layers of a corresponding multi-layer neural network. One or more additional layers can be added to map the top-level features $\hat{h}_k$ to the predictions associated with a target variable (here the probabilities associated with each class in a classification task). For more details on RBMs and DBNs, and a survey of related models and deep architectures, see Bengio (2009).

5.1.2 Stacked Denoising Auto-Encoders

The second model, by Vincent et al. (2008), is the so-called Stacked Denoising Auto-Encoder (SDAE). It borrows the greedy principle from DBNs, but uses denoising auto-encoders as a building block for unsupervised modeling. An auto-encoder learns an encoder $h(\cdot)$ and a decoder $g(\cdot)$ whose composition approaches the identity for examples in the training set, i.e., $g(h(\mathbf{x})) \approx \mathbf{x}$ for $\mathbf{x}$ in the training set.

Assuming that some constraint prevents $g(h(\cdot))$ from being the identity for arbitrary arguments, the auto-encoder has to capture statistical structure in the training set in order to minimize reconstruction error. However, with a high capacity code ($h(\mathbf{x})$ has too many dimensions), a regular auto-encoder could potentially learn a trivial encoding. Note that there is an intimate connection between minimizing reconstruction error for auto-encoders and contrastive divergence training for RBMs, as both can be shown to approximate a log-likelihood gradient (Bengio and Delalleau, 2009).

The *denoising auto-encoder* is a stochastic variant of the ordinary auto-encoder with the property that even with a high capacity model, it cannot learn the identity. Furthermore, its training criterion is a variational lower bound on the likelihood of a generative model. It is explicitly trained to denoise a corrupted version of its input. It has been shown on an array of datasets to perform significantly better than ordinary auto-encoders and similarly or better than RBMs when stacked into a deep supervised architecture (Vincent et al., 2008). Another way to prevent regular auto-encoders with more code units than inputs to learn the identity is to impose sparsity on the code (Ranzato et al., 2007, 2008).

We now summarize the training algorithm of the Stacked Denoising Auto-Encoders. More details are given by Vincent et al. (2008). Each denoising auto-encoder operates on its inputs $\mathbf{x}$, either the raw inputs or the outputs of the previous layer. The denoising auto-encoder is trained to reconstruct $\mathbf{x}$ from a stochastically corrupted (noisy) transformation of it. The output of each denoising auto-encoder is the "code vector" $h(\mathbf{x})$. In our experiments $h(\mathbf{x}) = \text{sigmoid}(\mathbf{b} + W\mathbf{x})$ is an ordinary neural network layer, with hidden unit biases $\mathbf{b}$, and weight matrix W. Let $C(\mathbf{x})$ represent a stochastic corruption of $\mathbf{x}$. As done by Vincent et al. (2008), we set $C_i(\mathbf{x}) = x_i$ or 0, with a random subset (of a fixed size) selected for zeroing. We have also considered a salt and pepper noise, where we select a random subset of a fixed size and set $C_i(\mathbf{x}) = \text{Bernoulli}(0.5)$. The "reconstruction" is obtained from the noisy input with $\hat{\mathbf{x}} = \text{sigmoid}(\mathbf{c} + W^T h(C(\mathbf{x})))$, using biases $\mathbf{c}$ and the transpose of the feed-forward weights W. In the experiments on images, both the raw input x_i and its reconstruction $\hat{x}_i$ for a particular pixel i can be interpreted as a Bernoulli probability for that pixel: the probability of painting the pixel as black at that location. We denote $\partial\text{KL}(\mathbf{x}||\hat{\mathbf{x}}) = \sum_i \partial\text{KL}(x_i||\hat{x_i})$ the sum of component-wise KL divergences between the Bernoulli probability distributions associated with each element of $\mathbf{x}$ and its reconstruction probabilities $\hat{\mathbf{x}}$: $\text{KL}(\mathbf{x}||\hat{\mathbf{x}}) = -\sum_i (x_i \log \hat{x}_i + (1 - x_i)\log(1 - \hat{x}_i))$. The Bernoulli model only makes sense when the input components and their reconstruction are in $[0, 1]$; another option is to use a Gaussian model, which corresponds to a Mean Squared Error (MSE) criterion.

With either DBN or SDAE, an output logitsic regression layer is added after unsupervised training. This layer uses softmax units to estimate $P(\text{class}|\mathbf{x}) = \text{softmax}_{\text{class}}(\mathbf{y}) = \frac{\exp y_{\text{class}}}{\sum_{j=1}^{N} \exp(y_j)}$, where $\mathbf{y}$ is the output of the network and N is the number of classes. The whole network is then trained as usual for multi-layer perceptrons, to minimize the output prediction error. In our experiments, we minimize the negative log-likelihood of the correct class given the raw input.

5.2 Datasets

We experimented on three datasets:

`MNIST` the digit classification dataset by LeCun et al. (1998), containing 60,000 training and 10,000 testing examples of 28x28 handwritten digits in grayscale.

`InfiniteMNIST` a dataset by Loosli et al. (2007), which is an extension of `MNIST` from which one can obtain a quasi-infinite number of examples. The samples are obtained by performing random elastic deformations of the original `MNIST` digits. In this dataset, there is only one set of examples, and the models will be compared by their (online) performance on it.

`Shapeset` is a synthetic dataset. The underlying task is binary classification of 10×10 images of triangles and squares. The examples show images of shapes with many variations, such as size, orientation and gray-level. The dataset is composed of 50000 training, 10000 validation and 10000 test images.

5.3 Setup

The models used are

1. Deep Belief Networks containing Bernoulli RBM layers
2. Stacked Denoising Auto-Encoders with Bernoulli input units and
3. standard feed-forward multi-layer neural networks

each with 1–5 hidden layers. Each hidden layer contains the same number of hidden units, which is a hyperparameter. The other hyperparameters are the unsupervised and supervised learning rates, the L_2 penalty / weight decay[3], and the fraction of stochastically corrupted inputs (for the SDAE). For `MNIST`, the number of supervised and unsupervised passes through the data (epochs) is 50 each. With `InfiniteMNIST`, we perform 2.5 million unsupervised updates followed by 7.5 million supervised updates[4]. The standard feed-forward networks are trained using 10 million supervised updates. For `MNIST`, model selection is done by choosing the hyperparameters that optimize the supervised (classification) error on the validation set. For `InfiniteMNIST`, we use the average online error over the last million examples for hyperparameter selection. In all cases, purely stochastic gradient updates are applied.

The experiments involve the training of deep architectures with a variable number of layers with and without pre-training. For a given layer, weights are initialized using random samples from uniform$[-1/\sqrt{k}, 1/\sqrt{k}]$, where k is the number of connections that a unit receives from the previous layer (the fan-in). Either supervised gradient descent or pre-training follows.

3. A term that penalizes $||\theta||_2$ is added to the supervised objective, where θ are the weights of the network
4. The number of examples was chosen to be as large as possible, while still allowing us to explore a variety of hyper-parameters

In most cases (for MNIST), we first launched a number of experiments using a cross-product of hyperparameter values[5] applied to 10 different random initialization seeds. We then selected the hyperparameter sets giving the best validation error for each combination of model (with or without pre-training), number of layers, and number of training iterations. Using these hyper-parameters, we launched experiments using an additional 400 initializations.

6. The Effect of Pre-training

We start by a presentation of large-scale simulations that were intended to confirm some of the previously published results about deep architectures. In the process of analyzing them, we start making connections to our hypothesis and motivate the experiments that follow.

6.1 Better Generalization

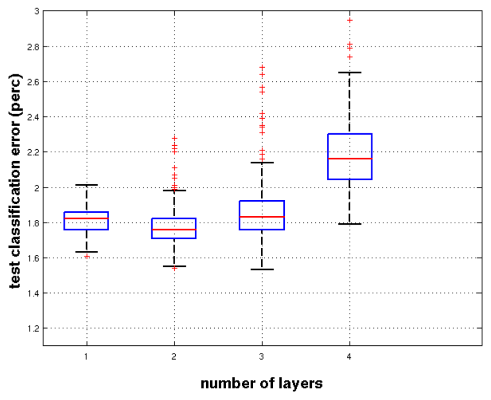
When choosing the number of units per layer, the learning rate and the number of training iterations to optimize classification error on the validation set, unsupervised pre-training gives substantially lower test classification error than no pre-training, for the same depth or for smaller depth on various vision datasets (Ranzato et al., 2007; Bengio et al., 2007; Larochelle et al., 2009, 2007) no larger than the MNIST digit dataset (experiments reported from 10,000 to 50,000 training examples).

Such work was performed with only one or a handful of different random initialization seeds, so one of the goals of this study was to ascertain the effect of the random seed used when initializing ordinary neural networks (deep or shallow) and the pre-training procedure. For this purpose, between 50 and 400 different seeds were used to obtain the graphics on MNIST.

Figure 1 shows the resulting distribution of test classification error, obtained with and without pre-training, as we increase the depth of the network. Figure 2 shows these distributions as histograms in the case of 1 and 4 layers. As can be seen in Figure 1, pre-training allows classification error to go down steadily as we move from 1 to 4 hidden layers, whereas without pre-training the error goes up after 2 hidden layers. It should also be noted that we were unable to effectively train 5-layer models without use of pre-training. Not only is the error obtained on average with pre-training systematically lower than without the pre-training, it appears also more robust to the random initialization. With pre-training the variance stays at about the same level up to 4 hidden layers, with the number of bad outliers growing slowly.

Contrast this with the case without pre-training: the variance and number of bad outliers grows sharply as we increase the number of layers beyond 2. The gain obtained with pre-training is more pronounced as we increase the number of layers, as is the gain in robustness to random initialization. This can be seen in Figure 2. The increase in error variance and mean for deeper architectures without pre-training suggests that **increasing depth increases the probability of finding poor local minima** when starting from random initialization. It is also interesting to note the low variance and small spread of errors

5. Number of hidden units $\in \{400, 800, 1200\}$; learning rate $\in \{0.1, 0.05, 0.02, 0.01, 0.005\}$; ℓ_2 cost penalty $\in \{10^{-4}, 10^{-5}, 10^{-6}, 0\}$; pre-training learning rate $\in \{0.01, 0.005, 0.002, 0.001, 0.0005\}$; corruption probability $\in \{0.0, 0.1, 0.25, 0.4\}$; tied weights $\in \{\text{yes}, \text{no}\}$.



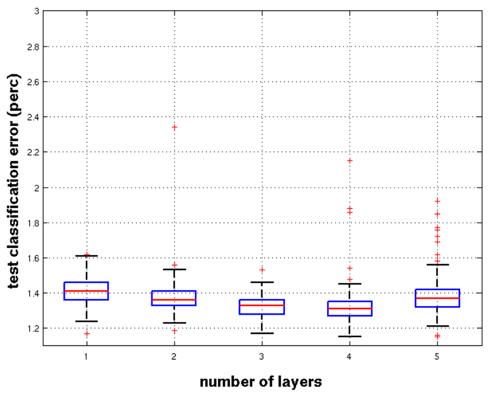


Figure 1: Effect of depth on performance for a model trained (**left**) without pre-training and (**right**) with pre-training, for 1 to 5 hidden layers (we were unable to effectively train 5-layer models without use of pre-training). Experiments on `MNIST`. Box plots show the distribution of errors associated with 400 different initialization seeds (top and bottom quartiles in box, plus outliers beyond top and bottom quantiles). Other hyperparameters are optimized away (on the validation set). *Increasing depth seems to increase the probability of finding poor local minima.*

obtained with 400 seeds with pre-training: it suggests that **pre-training is robust with respect to the random initialization seed** (the one used to initialize parameters before pre-training).

It should however be noted that there is a limit to the success of this technique: performance degrades for 5 layers on this problem. So while pre-training helps to increase the depth limit at which we are able to successfully train a network, it is certainly not the final answer.

These experiments show that the variance of final test error with respect to initialization random seed is larger without pre-training, and this effect is magnified for deeper architectures.

6.2 Visualization of Features

Figure 3 shows the weights (called filters) of the first layer of the DBN before and after supervised fine-tuning. For visualizing the 2nd and 3rd layer weights, we used the activation maximization technique described by Erhan et al. (2009): in essence, the method looks for the input pattern the maximizes the activation of a given unit. This is an optimization problem which is solved by performing gradient ascent in the space of the inputs, to find a local maximum of the activation function.

For comparison, we have also visualized the filters of a network for 1–3 layers in which no pre-training was performed (Figure 4). While the first layer filters do seem to correspond to localized features, 2nd and 3rd layers are not as interpretable anymore. Qualitatively

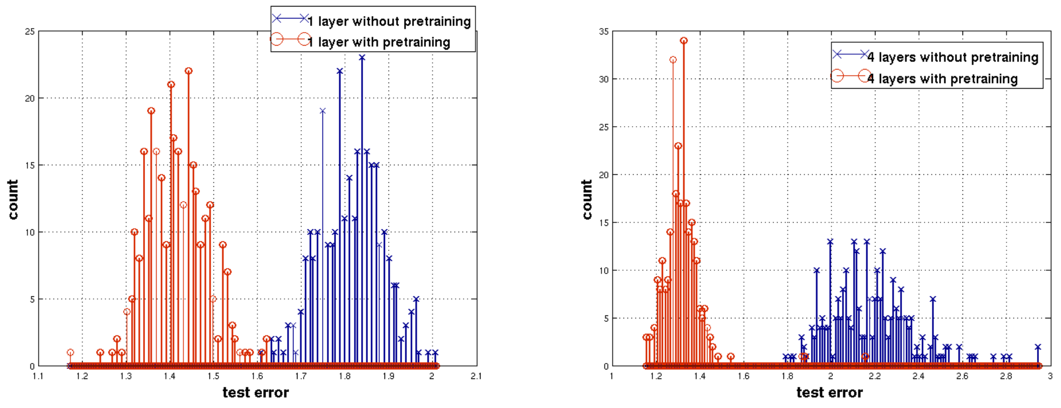


Figure 2: Histograms presenting the test errors obtained on `MNIST` using models trained with or without pre-training (400 different initializations each). **Left**: 1 hidden layer. **Right**: 4 hidden layers.

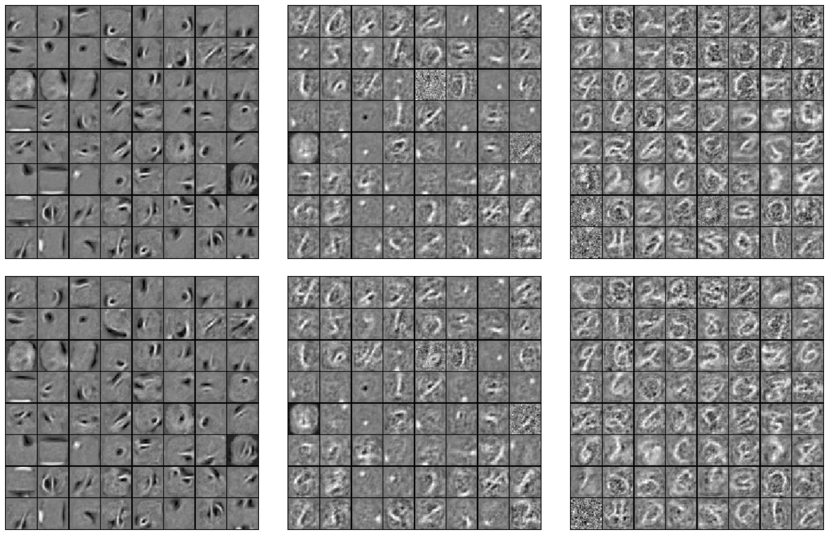


Figure 3: Visualization of filters learned by a DBN trained on `InfiniteMNIST`. The top figures contain a visualization of filters after pre-training, while the bottoms ones picture the same units after supervised fine-tuning; from left to right: units from the 1st, 2nd and 3rd layers, respectively.

speaking, filters from the bottom row of Figure 3 and those from 4 have little in common, which is an interesting conclusion in itself.

Several interesting conclusions can be drawn from this figure. First, unsupervised learning appears to make the network get "stuck", qualitatively speaking, in a certain region of weight space. Supervised learning, even with 7.5 million updates, does not change the weights in a significant way (at least visually). Second, different layers change differently: the first layer changes least, while supervised training has more effect on the 3rd layer. Such observations are consistent with the predictions made by our hypothesis: namely that the early dynamics of stochastic gradient descent, the dynamics induced by pre-training, can "lock" the training in a region of the parameter space that is essentially inaccessible for models that are trained in a purely supervised way.

Finally, the features increase in complexity as we add more layers. First layer weights seem to encode basic stroke-like detectors, second layer weights seem to detect digit parts, while top layer weights detect entire digits. The features are more complicated as we add more layers.

While Figures 3–4 shows only the filters obtained on `InfiniteMNIST`, the visualizations are similar when applied on `MNIST`. Likewise, the features obtained with SDAE result in qualitatively similar conclusions; Erhan et al. (2009) gives more details.

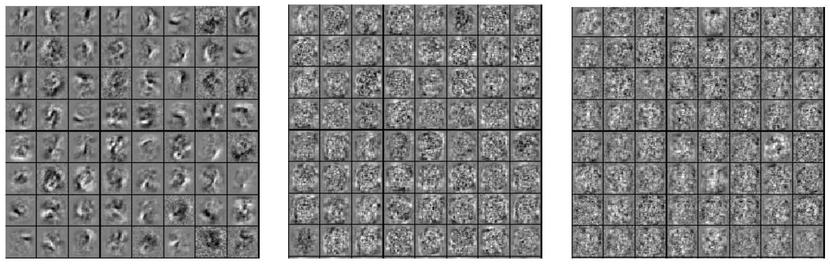


Figure 4: Visualization of filters learned by a network without pre-training, trained on `InfiniteMNIST`. The filters are shown after supervised training; from left to right: units from the 1st, 2nd and 3rd layers, respectively.

6.3 Visualization of Model Trajectories During Learning

Visualizing the learned features allows for a qualitative comparison of the training strategies for deep architectures. However it is not useful for investigating how these strategies deal with random initialization, as the features learned from multiple initializations look similar. If it was possible for us to visualize a variety of models at the same time, it would allow us to confirm our statements from the hypothesis, regarding the difference between pre-trained models and models that are initialized randomly. We could, for instance, confirm that the set of models that correspond to a pre-trained architecture with a certain set of parameters (but varying random seed) is distinct from the set of models that correspond to an architecture without pre-training, and that these two sets explore different regions of the parameter space, as predicted.

Unfortunately, it is not possible to directly compare parameter values of two architectures, because many permutations of the same parameters give rise to the same model. However, one can take a functional approximation approach in which we consider the outputs of each randomly initialized model as its representation in a high-dimensional space.

To visualize the trajectories followed in the landscape of the training criterion, we use the following procedure. For a given model, we compute all its outputs on the test set examples as one long vector summarizing where it stands in "function space". We get as many such vectors per model as passes over the training data. This allows us to plot many learning trajectories, one for each model (each associated with a different initialization seed), with or without pre-training. Using a dimensionality reduction algorithm we then map these vectors to a two-dimensional space for visualization[6]. Figure 5 and figure 6 present the results using dimensionality reduction techniques that focus respectively on local[7] and global structure[8]. Each point is colored according to the training iteration, to help follow the trajectory movement.

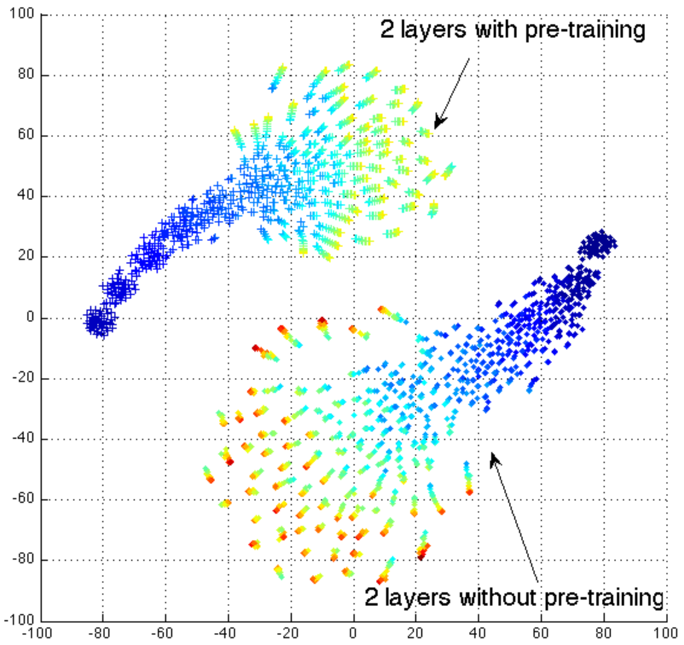


Figure 5: 2D visualizations with tSNE of the functions represented by 50 networks with and 50 networks without pre-training, as supervised training proceeds over MNIST. See section 6.3 for an explanation. Color from dark blue to yellow and red indicates a progression in training iterations (training is longer without pre-training). The plot shows models with 2 hidden layers but results are similar with other depths.

What seems to come out of these visualizations is the following:

6. Note that we can and do project the models with and without pre-training at the same time.
7. t-Distributed Stochastic Neighbor Embedding, or tSNE, by van der Maaten and Hinton (2008)
8. Isomap by Tenenbaum et al. (2000)

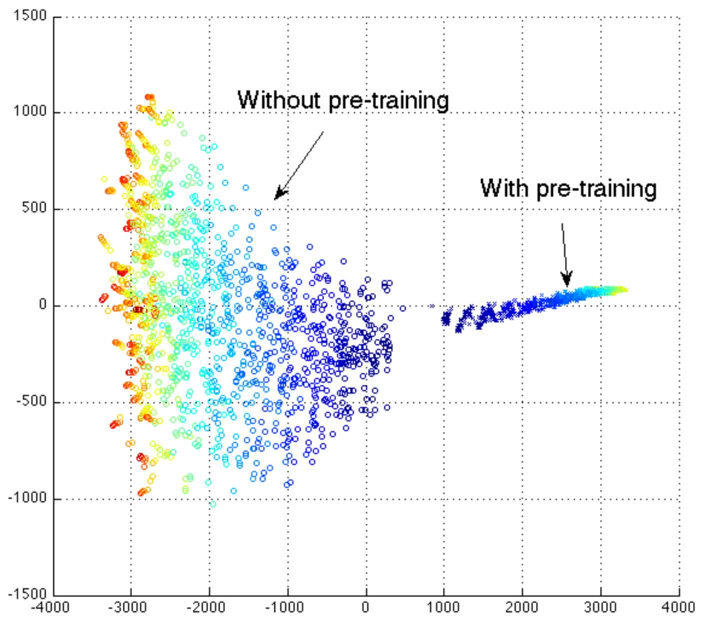


Figure 6: 2D visualization with ISOMAP of the functions represented by 50 networks with and 50 networks without pre-training, as supervised training proceeds over MNIST. See section 6.3 for an explanation. Color from dark blue to yellow and red indicates a progression in training iterations (training is longer without pre-training). The plot shows models with 2 hidden layers but results are similar with other depths.

1. The pre-trained and not pre-trained models start and *stay* in different regions of function space.
2. From the visualization focussing on local structure (Figure 5) we see that all trajectories of a given type (with pre-training or without) initially move togethers. However, at some point (after about 7 epochs) the different trajectories diverge (slowing down into elongated jets) and never get back close to each other. This suggests that each trajectory moves into a different local minimum[9].
3. From the visualization focussing on global structure (Figure 6), we see the pre-trained models live in a disjoint and much smaller region of space than the not pre-trained models. In fact, from the standpoint of the functions found without pre-training, the pre-trained solutions look all the same.

9. One may wonder if the divergence points correspond to a turning point in terms of overfitting. As shall be seen in below in Figure 8, the test error does not improve much after the 7th epoch, which reinforces this hypothesis.

The visualizations of the training trajectories do seem to confirm our suspicions. It is difficult to guarantee that each trajectory actually does end up in a different local minimum (corresponding to a different function and not only to different parameters). However, all tests performed (visual inspection of trajectories in function space, estimation of second derivatives in the directions of all the estimated eigenvectors of the Jacobian) are consistent with that interpretation.

We have also analyzed models obtained at the end of training, to visualize the training criterion in the neighborhood of the parameter vector θ^* obtained. This is achieved by randomly sampling a direction v (from the stochastic gradient directions) and by plotting the training criterion around θ^* in that direction, i.e. at $\theta = \theta^* + \alpha v$, for $\alpha \in \{-2.5, -2.4, \ldots, -0.1, 0, 0.1, \ldots 2.4, 2.5\}$, and v normalized ($||v|| = 1$). This analysis is visualized in Figure 7. The error curves look close to quadratic and we seem to be near a local minimum in all directions investigated, as opposed to a saddle point or a plateau. A more definite answer could be given by computing the full Hessian eigenspectrum, which would be expensive. Figure 7 also suggests that the error landscape is a bit flatter in the case of pre-training, and flatter for deeper architectures.

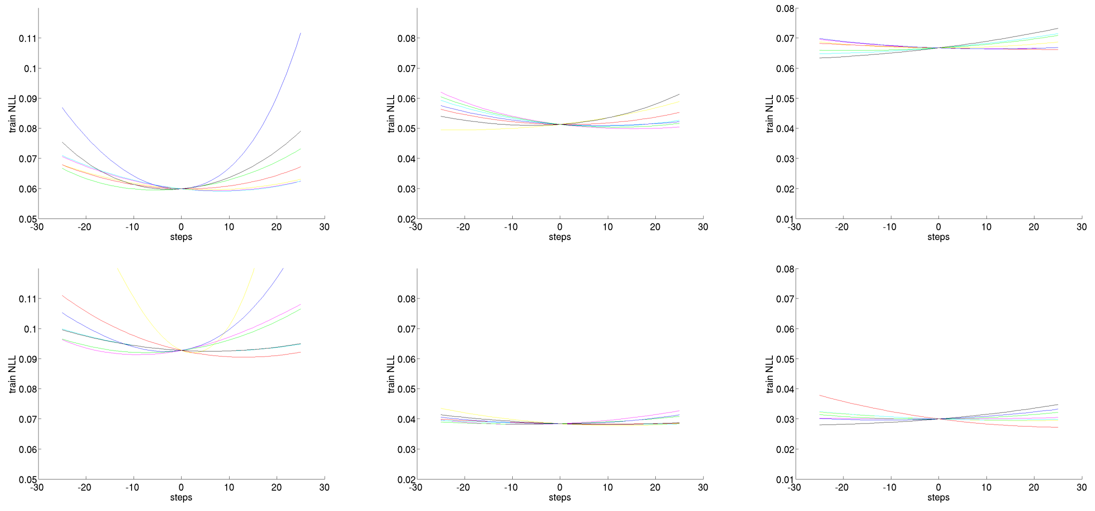


Figure 7: Training errors obtained on `Shapeset` when stepping in parameter space around a converged model in 7 random gradient directions (stepsize of 0.1). **Top**: no pre-training. **Bottom**: with pre-training. **Left**: 1 hidden layer. **Middle**: 2 hidden layers. **Right**: 3 hidden layers.

6.4 Implications

The series of results presented so far present a picture that is consistent with our hypothesis. Better generalization that seems to be robust to random initializations is indeed achieved by pre-trained models, which indicates that learning $P(X)$ via unsupervised is helpful in learning $P(Y|X)$. The error landscapes that we visualized point to the fact that there are many local minima. The pre-trained models seem to end up in distinct regions of these error landscapes (and, implicitly, in different parts of the parameter space). This is both seen from the functional space approximation trajectories and from the fact that the visualizations of the learned features are qualitatively very different from those obtained by model without pre-training.

7. The Role of Pre-training

The observations so far confirm that starting the supervised optimization from pre-trained weights rather than from random initialized weights consistently yields better performing classifiers on `MNIST`. To better understand where this advantage came from, it is important to realize that the *supervised objective being optimized is exactly the same in both cases*. The gradient-based optimization procedure is also the same. The only thing that differs is the starting point in parameter space: either picked at random or obtained after pre-training (which also starts from a random initialization).

Deep architectures, since they are built from the composition of several layers of non-linearities, yield an error surface that is non-convex and hard to optimize, with the suspected presence of many local minima (as also shown by the above visualizations). A gradient-based optimization should thus end in the local minimum of whatever *basin of attraction* we started from. From this perspective, the advantage of pre-training could be that it puts us in a region of parameter space where basins of attraction run deeper than when picking starting parameters at random. The advantage would be due to a better **optimization**.

Now it might also be the case that pre-training puts us in a region of parameter space in which training error is not necessarily better than when starting at random (or possibly worse), but which systematically yields better generalization (test error). Such behaviour would be indicative of a **regularization** effect.

Finally, a very simple explanation could be a most obvious one: namely the disparity in the magnitude of the weights at the start of the supervised training phase. We shall analyze (and rule out) this hypothesis first.

7.1 Experiment 1: Is Pre-training a Better Supervised Conditioning Process?

Typically gradient descent training of the deep model is initialized with randomly assigned weights, small enough to be in the linear region of the parameter space (close to zero for most neural network and DBN models). It is reasonable to ask if the advantage imparted by having an initial pre-training phase is simply due to the weights being larger and therefore somehow providing a better "conditioning" of the initial values for the optimization process; we wanted to rule out this possibility.

By conditioning, we mean the range and marginal distribution from which we draw initial weights. In other words, could we get the same performance advantage as pre-training if we were still drawing the initial weights independently, but form a more suitable distribution than the uniform$[-1/\sqrt{k}, 1/\sqrt{k}]$? To verify this, we performed pre-training, and computed marginal histograms for each layer's pre-trained weights and biases. We then resampled new "initial" random weights and biases according to these histograms, and performed fine-tuning from there.

Two scenarios can be imagined. In the first, the initialization from marginals leads to better performance than the standard initialization (when no pre-training is used). This would mean that pre-training does provide a better marginal conditioning of the weights. In the second scenario, the marginals lead to performance similar or worse to that without pre-training[10].

initialization.	Uniform	Histogram	Unsup.pre-tr.
1 layer	1.81 ± 0.07	1.94 ± 0.09	1.41 ± 0.07
2 layers	1.77 ± 0.10	1.69 ± 0.11	1.37 ± 0.09

Table 1: Effect of various initialization strategies on 1 and 2-layer architectures: independent uniform densities (one per parameter), independent densities from the marginals after pre-training, or unsupervised pre-training (which samples the parameters in a highly dependent way so that they collaborate to make up good denoising auto-encoders.) Experiments on `MNIST`, numbers are mean and std test errors.

What we observe in Table 1 falls within the first scenario. While it is true that initializing the weights to match the marginal distributions at the end of pre-training did seem to slightly improve the generalization error on MNIST, it is far from fully accounting for the discrepancy between the pre-trained and non-pre-trained results.

This experiment constitutes evidence against the preconditioning hypothesis, but does not exclude either the optimization hypothesis or the regularization hypothesis.

7.2 Experiment 2: The Effect of Pre-training on Training Error

The optimization and regularization hypotheses diverge on their prediction on how pre-training should affect the training error: the former predicts that pre-training should result in a lower training error, while the latter predicts the opposite. To ascertain the influence of these two possible explanatory factors, we looked at the test cost (Negative Log Likelihood on test data) obtained as a function of the training cost, along the trajectory followed in parameter space by the optimization procedure. Figure 8 shows 400 of these curves started from a point in parameter space obtained from random initialization, i.e. without pre-training (blue), and 400 started from pre-trained parameters (red).

The experiments were performed for networks with 1, 2 and 3 hidden layers. As can be seen in Figure 8, while for 1 hidden layer, pre-training reaches lower training cost than no pre-training, hinting towards a better optimization, this is not necessarily the case for the deeper networks. The remarkable observation is rather that, *at a same training cost level, the pre-trained models systematically yield a lower test cost* than the randomly initialized ones. The advantage appears to be one of *better generalization rather than merely a better optimization procedure.*

10. We observed that the distribution of weights after unsupervised pre-training is fat-tailed. It is conceivable that sampling from such a distribution in order to initialize a deep architecture could actually *hurt* the performance of a deep architecture (compared to random initialization from a uniform distribution), since the fat-tailed distribution allows for configurations of initial weights, which are unlikely to be learned by unsupervised pre-training, because large weights could be sampled independently

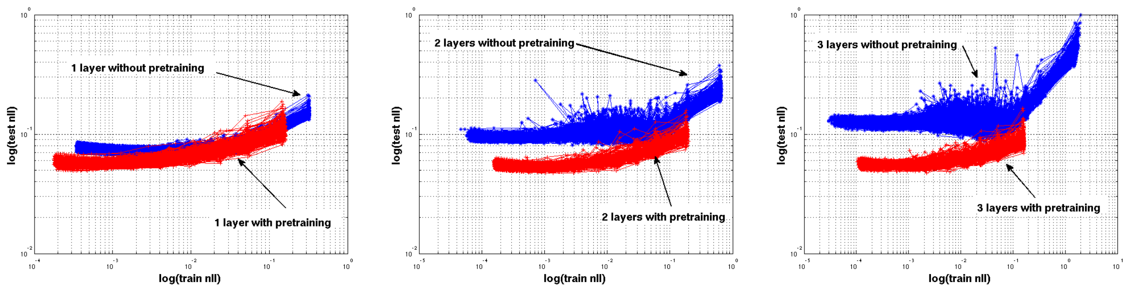


Figure 8: Evolution without pre-training (blue) and with pre-training (red) on `MNIST` of the log of the test NLL plotted against the log of the train NLL as training proceeds. Each of the 2×400 curves represents a different initialization. The errors are measured after each pass over the data. The rightmost points were measured after the first pass of gradient updates. Since training error tends to decrease during training, the trajectories run from right (high training error) to left (low training error). Trajectories moving up (as we go leftward) indicate a form of overfitting. All trajectories are plotted on top of each other.

This brings us to the following result: pre-training appears to have a similar effect to that of a good regularizer or a good "prior" on the parameters, even though no explicit regularization term is apparent in the cost being optimized. As we stated in the hypothesis, it might be reasoned that restricting the possible starting points in parameter space to those that minimize the pre-training criterion (as with the SDAE), does in effect restrict the set of possible final configurations for parameter values. Like regularizers in general, pre-training (in this case, with denoising auto-encoders) might thus be seen as decreasing the variance and introducing a bias[11]. Unlike ordinary regularizers, pre-training does so in a data-dependent manner.

7.3 Experiment 3: The Influence of the Layer Size

Another signature characteristic of regularization is that the effectiveness of regularization increases as the number of units increases; effectively trading off one constraint on the model complexity for another. In this experiment we explore the relationship between the number of units per layer and the effectiveness of pre-training. The hypothesis that pre-training acts as a regularizer would suggest that we should see a trend of increasing effectiveness of pre-training as the number of units per layer are increased.

The alternative optimization hypothesis would in fact predict the opposite result as the extra constraints imposed by limiting the number of units per layer renders the optimization problem more challenging Therefore we would expect to see a trend of increasing effectiveness of the pre-training optimization strategy as the number of units per layer are decreased.

11. towards parameter configurations suitable for performing denoising

We trained models on `MNIST` with and without pre-training using increasing layer sizes: 25, 50, 100, 200, 400, 800 units per layer. Results are shown in Figure 9. Qualitatively similar results were obtained on *Shapeset*. In the case of SDAE, we were expecting the denoising pre-training procedure to help classification performance most for large layers; this is because the denoising pre-training allows useful representations to be learned in the over-complete case, in which a layer is larger than its input (Vincent et al., 2008). What we observe is a more systematic effect: while pre-training helps for larger layers and deeper networks, it also appears to hurt for too small networks.

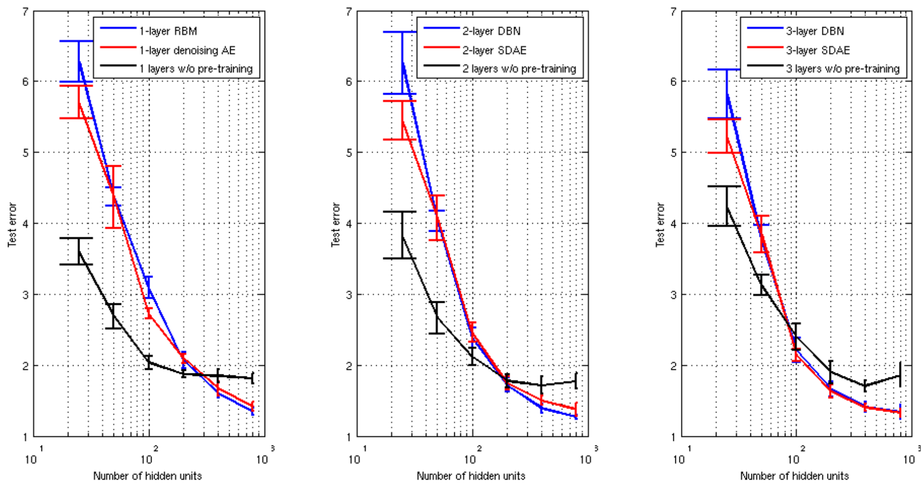


Figure 9: Effect of layer size on the changes brought by pre-training, for networks with 1, 2 or 3 hidden layers. Experiments on `MNIST`. Error bars have a height of two standard deviations (over initialization seed). Pre-training hurts for smaller layer sizes and shallower networks, but it helps for all depths for larger networks.

Figure 9 also shows that DBNs behave qualitatively like SDAEs, in the sense that pre-training architectures with smaller layers hurts performance. Experiments on `InfiniteMNIST` reveal results that are qualitatively the same. Such an experiment seemingly points to a re-verification of the regularization hypothesis. In this case, it would seem that pre-training acts as an additional regularizer for both DBN and SDAE models—on top of the regularization provided by the small size of the hidden layers. With excessive regularization, generalization is hurt, and it is hurt more with unsupervised pre-training presumably because of the extra regularization effect: small networks have a limited capacity already so further restricting it (or introducing an additional bias) can harm generalization. Such a result seems incompatible with a pure optimization effect.

We also obtain the result that DBNs and SDAEs seem to have qualitatively similar effects as pre-training strategies.

7.4 Experiment 4: Challenging the Optimization Hypothesis

Experiments 1–3 results are consistent with the regularization hypothesis and Experiments 2–3 would appear to directly support the regularization hypothesis over the reasonable alternative – the hypothesis that pre-training aids in optimizing the deep model objective function.

In the literature there is some support for the optimization hypothesis. Bengio et al. (2007) constrained the top layer of a deep network to have 20 units and measured the training error of networks with and without pre-training. The idea was to prevent the networks from overfitting the training error simply with the top hidden layer, thus to make it clearer whether some optimization effect (of the lower layers) was going on. The reported error was lower for pre-trained networks. One problem with the experimental paradigm used by Bengio et al. (2007) is their use of early-stopping. This is problematic because, as previously mentioned, early stopping is itself a regularizer, and it can influence greatly the training error that is obtained; it is entirely possible that if Bengio et al. (2007) had run the models to convergence, the results would have been different.

Figure 10 shows what happens without early stopping. The training error is still higher for pre-trained networks even though the generalization error is lower. This result now favors the regularization hypothesis against the optimization story. What may have happened is that early stopping prevented the networks without pre-training from moving towards too much towards their local minimum.

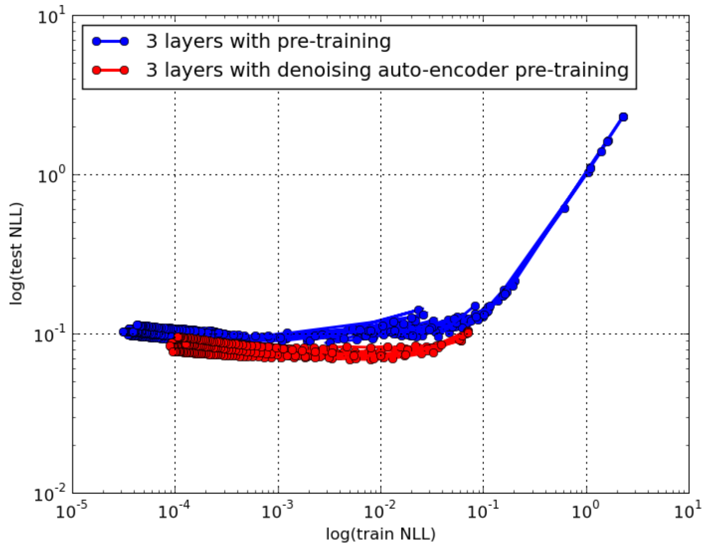


Figure 10: For MNIST, a plot of the log(train NLL) vs. log(test NLL) at each epoch of training. The top layer is constrained to 20 units.

7.5 Experiment 5: Comparing pre-training to L_1 and L_2 regularization

An alternative hypothesis would be that classical ways of regularizing could perhaps achieve the same effect as pre-training. We investigated the effect of L_1 and L_2 weight regularization (i.e. adding a $||\theta||_1$ or $||\theta||_2$ term to the supervised objective function) of a network without pre-training. We found that while in case of `MNIST` a small penalty can in principle help, the gain is nowhere near as large as it is with pre-training. For `InfiniteMNIST`, the optimal amount of L_1 and L_2 regularization is zero[12].

This is not an entirely surprising finding: not all regularizers are created equal and these results are consistent with the literature on semi-supervised training that show that unsupervised training can be a particularly effective form of regularization.

7.6 Summary of Findings: Experiments 1-5

So far, the results obtained from the previous experiments point towards a pretty clear explanation of the effect of pre-training: namely, that its effect is a regularization effect. We have seen that it is not simply sufficient to sample random weights with the same magnitude: the (data-dependent) unsupervised initialization is crucial. We have also observed that canonical regularizers (L_1/L_2 penalties on the weights) do not achieve the same level of perfomance.

The most compelling pieces of evidence in support of the regularization hypothesis are Figures 8 and 9. The reasonable alternative explanation—that pre-training has an optimization effect—suggested by Bengio et al. (2007) doesn't seem to be supported by the proper experimental results. But this isn't the end of the story, as the next batch of experiments shows.

8. The Online Learning Setting

Our hypothesis included not only the statistical/phenomenological hypothesis that pre-training acted as a regularizer, but also contains a mechanism for how such behaviour arises both as a consequence of the dynamic nature of training—following a stochastic gradient through two phases of training and as a consequence of the non-convexity of the supervised objective function.

In our hypothesis, we posited that early examples induce changes in the magnitude of the weights that increase the amount of non-linearity of the network, which in turn decreases the number of regions accessible to the stochastic gradient descent procedure. This means that the early examples determine the basin of attraction for the remainder of training; this also means that the early examples have a (perhaps disproportionate) influence on the configuration of parameters that the models take.

According to the hypothesized mechanism, we would predict that in the online learning setting with unbounded or very large datasets, the behaviour of pre-training would diverge from the behaviour of a canonical regularizer (L_1/L_2). This is because the effectiveness of a canonical regularizer **decreases** as the dataset grows, whereas the effectiveness of pre-training as a regularizer is **maintained** as the dataset grows.

12. Which is consistent with the classical view of regularization, in which its effect should diminish as we add more and more data.

In this section we empirically challenge this aspect of the hypothesis and show that the evidence does indeed support our hypothesis over a more standard or expected regularization behavior.

8.1 Experiment 6: Effect of Pre-training with Very Large Datasets

The next set of results point are perhaps the most surprising findings of this paper. Figure 11 shows the online classification error (on the next block of examples, as a moving average) for 6 architectures that are trained on `InfiniteMNIST`: 1 and 3-layer DBNs, 1 and 3-layer SDAE, as well as 1 and 3-layer networks without pre-training. Note that stochastic gradient descent in online learning is a stochastic gradient descent optimization of the generalization error, so good online error in principle implies that we are optimizing well the training criterion.

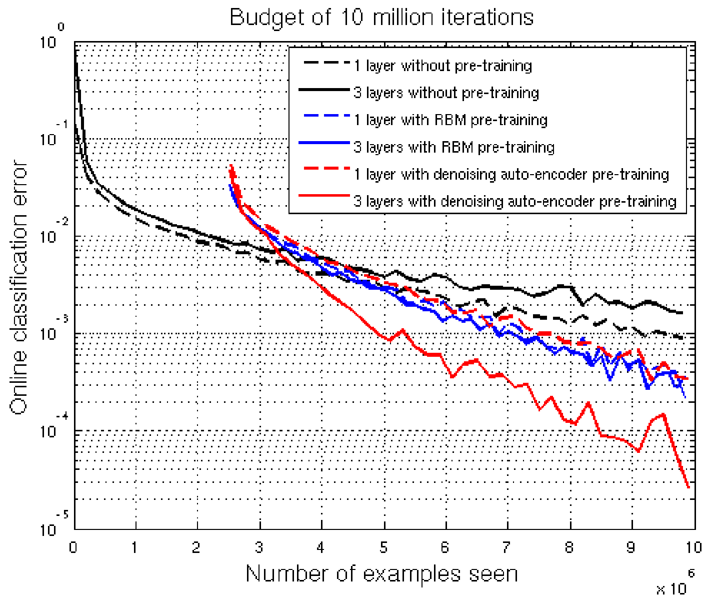


Figure 11: Comparison between 1 and 3-layer networks trained on `InfiniteMNIST`.

We can draw several observations from these experiments. First, 3-layer networks without pre-training are worse at generalization, compared to the 1-layer equivalent. This confirms the hypothesis that even in an online setting, optimization of deep networks is harder than shallow ones. Second, 3-layer SDAE models seem to generalize better than 3-layer DBNs. Finally and most importantly, the pre-training advantage does not vanish as the number of training examples increases, on the contrary. This seems to support the hypothesis related to the optimization effect of pre-training.

Note that the number of hidden units of each model is a hyperparameter. So theoretical results suggest that 1-layer networks without pre-training should in principle be able to represent the input distribution as capacity and data grow. Instead, without pre-training, the networks are not able to take advantage of the additional capacity, which again points towards the optimization explanation. It is clear, however, that **the starting point of**

the non-convex optimization matters, even for networks that are seemingly "easier" to optimize (1-layer ones), which supports our hypothesis.

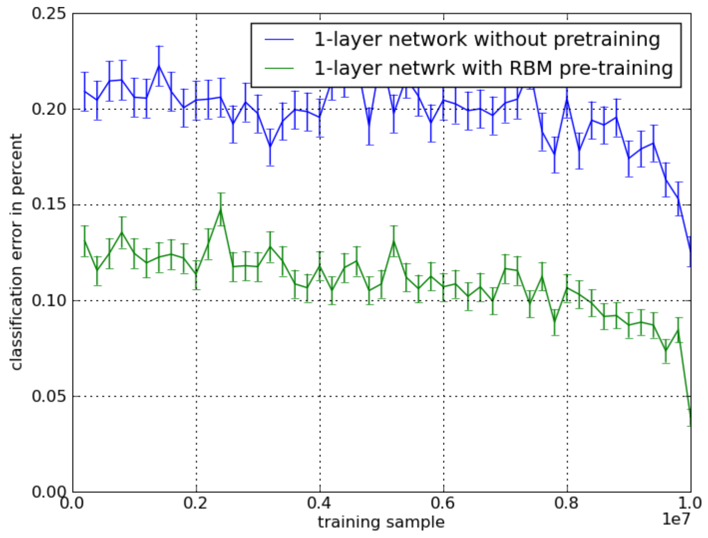


Figure 12: Error of 1-layer network with RBM pre-training and without, on the 10 million examples used for training it. The errors are calculated in the same order as the examples were presented during training.

Another experiment that shows the effects of large-scale online stochastic non-convex optimization is shown in Figure 12. In the setting of `InfiniteMNIST`, we compute the error on the *training set*, in the same order that we presented the examples to the models. We observe several interesting results: first, note that both models are better at classifying examples that are closest to the last examples seen. This is a natural effect of stochastic gradient descent. Note also that examples at the beginning of training are essentially like test examples for both models. Finally, we observe that the pre-trained model is better across the board *on the training set*. This fits well with the optimization hypothesis, since it shows that pre-training has an optimization effect.

What happens in this setting is that the training and generalization errors converge as the empirical distribution (defined by the training set) converges to the true data distribution. These results show that the effectiveness of pre-training does not diminish with increasing dataset sizes. This would be unexpected from a superficial understanding of pre-training as a regularization method. However it is entirely consistent with our interpretation, stated in our hypothesis, of the role of pre-training in the online setting with stochastic gradient descent training on a non-convex objective function.

8.2 Experiment 7: The Effect of Example Ordering

The hypothesized mechanism implies, due to the dynamics of learning—the increase in magnitude and non-linearity as training proceeds, as well as the dependence of the basin of attraction on early data—that, when training with very large data sets, we should see increased sensitivity to early examples. In the case of `InfiniteMNIST` we operate in an online stochastic optimization regime, where we try to find a local minimum of a highly non-convex objective function. It is then interesting to study to what extent the outcome of this optimization is influenced by the examples seen at different points during training, and whether the early examples have a stronger influence (which would not be the case in the convex case).

To quantify the variance of the outcome and to compare these variances for models with and without pre-training, we proceeded with the following experiment: given a dataset with 10 million examples, we vary the first million examples (across 10 different random draws, sampling a different set of 1 million examples each time) and keep the other ones fixed. After training the (10) models, we measure the variance of the *output* of the networks on a fixed test set (i.e. we measure the variance in function space). We then vary the next million examples in the same fashion, and so on, to see how much each of the ten parts of the training set influenced the final function.

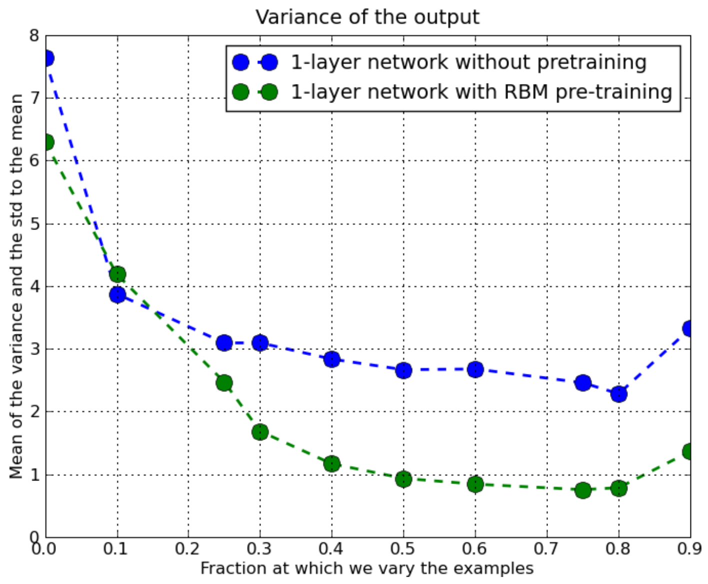


Figure 13: Variance of the output of a trained network with 1 layer. The variance is computed as a function of the point at which we vary the training samples. Note that the 0.25 mark corresponds to the start of pre-training.

Figure 13 shows the outcome of such an analysis. The samples at the beginning[13] do seem to influence the output of the networks more than the ones at the end. However, this variance is *lower* for the networks that have been pre-trained. In addition to that, one should note that the variance of the pre-trained network at 0.25 (i.e. the variance of the output as a function of the first samples used for supervised training) is *lower* than the variance of the supervised network at 0.0. Such results imply that unsupervised pre-training can be

13. Which are *unsupervised* examples, for the green curve, until 0.25.

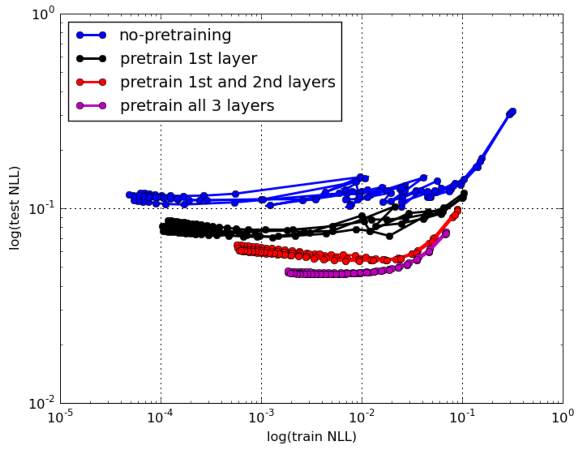
seen as a sort of variance reduction technique, consistent with a regularization hypothesis. Finally, both networks have higher output variances as a function of the *last examples* used for optimization, which is simply due to the fact that we operate in a stochastic gradient regime where the most recent examples' gradient has a greater effect, generally speaking.

These results are consistent with what our hypothesis predicts: both the fact that early examples have greater influence (i.e. the variance is higher) and that pre-trained models seem to reduce this variance are in agreement with what we would have expected.

8.3 Experiment 8: Pre-training only k layers

From Figure 11 we can see that pre-training makes quite a difference for 3 layers, on `InfiniteMNIST`. In Figure 14 we explore the link between depth and pre-training in more detail. The setup is as follows: for both `MNIST` and `InfiniteMNIST` we pre-train only the bottom k layers and randomly initialize the top $n-k$ layers in the usual way. In this experiment, $n = 3$ and we vary k from 0 (which corresponds to a network with no pre-training) to $k = n$ (which corresponds to the normal pre-trained case).

For `MNIST`, we plot the log(train NLL) vs. log(test NLL) trajectories, where each point corresponds to a measurement after a certain number of epochs. The trajectories go roughly from the right to left and from top to bottom, corresponding to the lowering of the training and test errors. We can also see that models overfit from a certain point onwards.



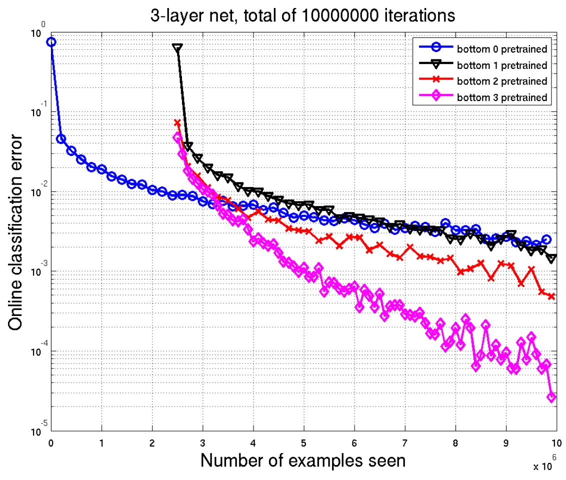


Figure 14: *On the left*: for `MNIST`, a plot of the log(train NLL) vs. log(test NLL) at each epoch of training. We pre-train the first layer, the first two layers and all three layers using RBMs and randomly initialize the other layers; we also compare with the network whose layers are all randomly initialized. *On the right*: `InfiniteMNIST`, the online classification error. We pre-train the first layer, the first two layers or all three layers using denoising auto-encoders and leave the rest of the network randomly initialized.

For InfiniteMNIST, we simply show the online error. The results are ambiguous w.r.t. our hypothesis, regarding the difficulty of optimizing the lower layers versus the the higher ones. We would have expected that the largest incremental benefit came from pre-training the first layer or first two layers. It is true for the first two layers, but not the first. Note that the log-scale (on the right) induces a distortion which makes the improvement of pre-training the third layer appear larger, where we are already near zero generalization error. As we pre-train more layers, the models become better at generalization. In the case of the finite MNIST, note how the final training error (after the same number of epochs) becomes *worse* with pre-training of more layers. This clearly brings additional support to the regularization explanation.

9. Discussion and Conclusions

We have shown that pre-training adds robustness to a deep architecture. The same set of results also suggests that increasing the depth of an architecture that is not pre-trained increases the probability of finding poor local minima. Pre-trained networks give consistently better generalization. Our visualizations point to the observations that pre-trained networks learn qualitatively different features (if networks are visualized in the weight space) compared to networks without pre-training. Moreover, the trajectories of networks with different initialization seeds seem to fall into many distinct local minima, which are again different (and seemingly far apart) depending on whether we use pre-training or not.

We have shown that pre-training is not simply a way of getting a good initial marginal distribution, and that it captures more intricate dependencies between parameters. One of our findings is that deep networks with pre-training seem to exhibit some properties of a regularizer: with small enough layers, pre-trained deep architectures are systematically worse that randomly initialized deep architectures. Moreover, when the layers are big enough, the pre-trained models obtain worse training errors, but better generalization performance. Additionally, we have re-done an experiment which purportedly showed that pre-training can be explained with with an optimization hypothesis and observed a regularization effect instead. We isalso shown that classical regularization techniques (such as L_1/L_2 penalties on the network weights) cannot achieve the same performance as pre-training, so if pre-training is a regularizer, it is certainly be a rather different kind.

The two pre-training strategies considered—denoising auto-encoders and Restricted Boltzmann Machines—seem to produce qualitative similar observations. We have observed that, surprisingly, the pre-training advantage is present even in the case of really large training sets, pointing towards the conclusion that the starting point in a non-convex optimization problem is indeed quite important; a fact confirmed by our visualizations of filters at various levels in the network. Finally, the other important set of results show us that pre-training acts like a variance reduction technique, yet a network with pre-training has a lower training error on a very large dataset, which supports an optimization interpretation of the effect of pre-training.

How do we make sense of all these results? The contradiction between what looks like regularization effects and what looks like optimization effects appears, on the surface, unresolved. Instead of sticking to these labels, we attempted to draw a hypothesis, described in Section 3 about the dynamics of learning in an architecture that is trained using two phases (pre-training and supervised fine-tuning), which we believe to be consistent with all the above results.

This hypothesis suggests that there are consequences of the non-convexity of the supervised objective function, which we observed in various ways throughout our experiments. One of these consequences is that early examples have a big influence on the outcome of training and this is one of the reasons why in a large-scale setting the influence of pre-training is still present. Throughout this paper, we have delved on the idea that the basin of attraction induced by the early examples (in conjuction with pre-training) is, for all purposes, a basin from which supervised training does not escape.

This effect can be observed from the various visualizations and performance evaluations that we made. *Pre-training,* **as a regularizer** *that only influences the starting point of supervised training, has an effect that, contrary to classical regularizers, does not disappear* (at least as far as we can show from our results).

One of the main messages that our results imply is that the optimization of a non-convex objective function with stochastic gradient descent presents challenges for analysis, especially in a regime with large amounts of data. Our analysis so far shows that it is possible for networks that are trained in such a regime to be influenced more by early examples. This can pose problems in scenarios where we would like our networks to be able to move to other basins of attraction during training.

One interesting realization is that with a small training set, we do not usually put a lot of importance on minimizing the training error, because overfitting is a major issue; the training error is not a good way to distinguish between the generalization performance of two models. In that setting, pre-training helps to find local minima that have better generalization error. With a large training set, as we saw in Figure 12, the empirical and true distributions converge. In such a scenario, *finding a better local minimum will matter and stronger (better) optimization strategies should have a significant impact on generalization when the training set is very large.*

More work is certainly needed in order to better understand these effects. While our results are relatively comprehensive as far as MNIST is concerned, they are limited in that we only considered `MNIST` and its extended variation, `InfiniteMNIST`. Our original goal was to have well-controlled experiments with well understood datasets. It was not to advance a particular algorithm but rather to try to better understand a phenomenon that has been well documented elsewhere. The use of toy datasets allows us to better control the experiments in ways that would not otherwise be possible. Future work would certainly be about reproducing such results on other datasets, especially truly large-scale data.

Our results suggest that optimization in deep networks is a complicated problem that is influenced in part by the early examples during training. Future work should clearly test this hypothesis. If it is true and we want our learners to capture really complicated distributions from very large training sets, it may mean that we should consider learning algorithms that reduce the effect of the early examples, allowing to escape from the attractors in which current learning dynamics get stuck.

Other avenues for future work include the analysis and understanding of deep semi-supervised techniques where one does not separate between the pre-training phase and the supervised phase, such as work by Weston et al. (2008) and Larochelle and Bengio (2008). Such algorithms fall more squarely into the realm of semi-supervised methods. We expect that analyses similar to the ones we performed would be potentially harder, but perhaps revealing as well.

Understanding and improving deep architectures remains a challenge. Our conviction is that devising improved strategies for learning in deep architectures requires a more profound understanding of the difficulties that we face with them. This work help with such understanding via extensive simulations and puts forward a hypothesis explaining the mechanisms behind pre-training, which is well supported by our results.

Acknowledgements

This research was supported by funding from NSERC, MITACS, FQRNT, and the Canada Research Chairs.

References

Shun-ichi Amari, Noboru Murata, Klaus-Robert Müller, Michael Finke, and Howard Hua Yang. Asymptotic statistical theory of overtraining and cross-validation. *IEEE Transactions on Neural Networks*, 8(5):985–996, 1997.

Andrew E. Barron. Complexity regularization with application to artificial neural networks. In G. Roussas, editor, *Nonparametric Functional Estimation and Related Topics*, pages 561–576. Kluwer Academic Publishers, 1991.

M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, to appear, 2009.

Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6):1601–1621, 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, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 107–114. MIT Press, Cambridge, MA, 2006.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

Marc H. Bornstein. *Sensitive periods in development : interdisciplinary perspectives / edited by Marc H. Bornstein*. Lawrence Erlbaum Associates, Hillsdale, N.J. :, 1987.

O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

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

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, Université de Montréal, 2009.

Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Proc. Intelligent Robots and Systems (IROS'08)*, 2008.

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.

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

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

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

Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted Boltzmann machines. In Andrew McCallum and Sam Roweis, editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 536–543. Omnipress, 2008.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Zoubin Ghahramani, editor, *ICML 2007: Proceedings of the Twenty-fourth International Conference on Machine Learning*, pages 473–480. Omnipress, 2007.

Hugo Larochelle, Yoshua Bengio, Jerome Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

Julia A. Lasserre, Christopher M. Bishop, and Thomas P. Minka. Principled hybrids of generative and discriminative models. In *CVPR '06: Proceedings of the 2006 IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition, pages 87–94, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. doi: http://dx.doi.org/10.1109/CVPR.2006.227.

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

Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area V2. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Léon Bottou and Michael Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, Montreal, June 2009. Omnipress.

Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. In Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007. URL `http://leon.bottou.org/papers/loosli-canu-bottou-2006`.

Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep learning from temporal coherence in video. In Léon Bottou and Michael Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 737–744, Montreal, June 2009. Omnipress.

Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2001.

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

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*, pages 1137–1144. MIT Press, 2007.

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*, Cambridge, MA, 2008. MIT Press.

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

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey E. Hinton. Restricted Boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM.

J. Sjöberg and L. Ljung. Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62(6):1391–1407, 1995.

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 high-dimensional data using t-sne. *Journal of Machine Learning Research*, 2008.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML 2008: Proceedings of the Twenty-fifth International Conference on Machine Learning*, pages 1096–1103, 2008.

Max Welling, Michal Rosen-Zvi, and Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, 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.

Long Zhu, Yuanhao Chen, and Alan Yuille. Unsupervised learning of probabilistic grammar-markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):114–128, 2009.