Why Does Unsupervised Pre-training Help Deep Learning?

Dumitru Erhan **Aaron Courville** **Yoshua Bengio** **Pascal Vincent**
DIRO, Université de Montréal
2920 chemin de la Tour
Montréal, Québec, H3T 1J8, Canada
first.last@umontreal.ca

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 being obtained in several areas, mostly on vision and language datasets. The best results obtained on supervised learning tasks often involve an unsupervised learning component, usually in an unsupervised pre-training phase. The main question investigated here is the following: why does unsupervised pre-training work so well? Through extensive experimentation, we explore several possible explanations discussed in the literature including its action as a regularizer (Erhan et al., 2009b) and as an aid to optimization (Bengio et al., 2007). Our results build on the work of Erhan et al. (2009b), showing that unsupervised pre-training appears to play predominantly a regularization role in subsequent supervised training. However our results in an online setting, with a virtually unlimited data stream, point to a somewhat more nuanced interpretation of the roles of optimization and regularization in the unsupervised pre-training effect.

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. Theoretical results (Yao, 1985; Håstad, 1986; Bengio et al., 2006), reviewed 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*. Recent theoretical and empirical work in statistical machine learning has suggested developing learning algorithms for these deep architectures, i.e., function classes obtained by composing multiple non-linear transformations. Bengio (2009) gives a comprehensive review.

Searching the parameter space of deep architectures is a difficult task because the training criterion is non-convex and involves many local minima. This was clearly demonstrated in recent empirical work (Erhan et al., 2009b) showing consistently that with hundreds of different random initializations, gradient descent converged each time to a different apparent local minimum, with solutions obtained from random initialization and purely supervised training consistently getting worse for architectures with more than 2 or 3 levels. This points to why, until recently, deep architectures have received little attention in the machine learning literature.

The breakthrough for deep architectures came in 2006 with the algorithms for training Deep Belief Networks (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 non-linear transformation of its input (the output of the previous layer) that captures the main variations in its input. This unsupervised pre-training only 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.

The objective of this paper is not to demonstrate a new learning algorithm for deep architectures but rather to shed some light on the existing algorithms through extensive simulations. We are interested in investigating why pre-training works and why pre-trained networks work so much better than networks trained in a traditional way. There are several reasonable hypotheses, several of which are explored in this paper. A first explanation, suggested by Er-

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

han et al. (2009b), is that of pre-training as a conditioning mechanism for the parameters of the network. An alternative, suggested by Bengio et al. (2007) is that pre-training is useful for initializing the network in a region of the parameter space where optimization is somehow easier (and where a better local optimum of the training criterion is found).

The results presented in this paper indicate that pre-training is a kind of *regularization* mechanism, by minimizing variance and introducing a bias towards configurations of the parameter space that are useful for unsupervised learning. Such a conclusion is also supported by the results obtained by Erhan et al. (2009b). Collectively, these findings place the recent advances in training deep architectures well within the realm of semi-supervised learning methods. Our simulations show, however, that pre-trained networks are unique among semi-supervised methods in that the unsupervised component is used purely as an initialization and that its beneficial effects do not appear to diminish as we perform supervised training.

2 Experimental Results in the Literature

We briefly review some of the more relevant experimental results from the literature.

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 (comparing 1,2,3,4 and 5 hidden layers) on various vision datasets (Ranzato et al., 2007; Bengio et al., 2007; Larochelle et al., 2009; Larochelle et al., 2007; Erhan et al., 2009b) no larger than the `MNIST` digit dataset (experiments reported from 10,000 to 50,000 training examples). See Erhan et al. (2009b) for the most comprehensive of experiments.

Aid to optimization. In these experiments, the training error of the trained classifiers is low (nearly 0 on `MNIST`) in all cases, with or without pre-training. Such a result would make it difficult to distinguish between the optimization and regularization effects of pre-training. Bengio et al. (2007) hypothesized that higher layers in the network were overfitting the training error, thus to make it clearer whether some optimization effect (of the lower layers) was going on, they constrained the top layer to be small (20 units instead of 500 and 1000). In that experiment, they show that the final training errors are *higher* without pre-training.

Distinct local minima. With 400 different random initializations, with or without pre-training, each trajectory ends up in a different apparent local minimum corresponding not only to different parameters but to a different function (Erhan et al., 2009b). It is difficult to guarantee that these are indeed local minima but 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. The regions in function spaced reached without pre-training and with pre-training seem completely disjoint (i.e. no model without pre-training ever gets close to a model with pre-training).

Lower variance. In the same set of experiments, the variance of final test error with respect to the initialization random seed is larger without pre-training, and this effect is magnified for deeper architectures (Erhan et al., 2009b). This supports a regularization explanation, but does not exclude an optimization hypothesis either.

Capacity control. Finally, when all the layers are constrained to a smaller size, the pre-training advantage disappears, and for very small sizes generalization is worse with pre-training (Erhan et al., 2009b). Such a result is highly compatible with a regularization effect and seems incompatible with a pure optimization effect.

3 Pre-training as a Regularizer

The results presented so far point to the following:

1. The supervised training criterion for a deep neural network is fraught with local minima, even more so than in an ordinary single-layer network.
2. Random initialization of a deep architecture falls with very high probability in the basin of attraction of a poor local minimum.
3. Unsupervised pre-training initializes a deep architecture in a basin of attraction of gradient descent corresponding to better generalization performance.

We are interested in finding an explanatory hypothesis that agrees with the above statements, is consistent with experimental results, provides us insight into pre-training as well as its effects and demystifies several aspects of deep architectures that are still unclear.

We noted already that Bengio et al. (2007) suggests that unsupervised pre-training is useful because it helps to better optimize the training criterion. As we will discuss below, results of Erhan et al. (2009b) seem to contradict it, pointing instead to the the idea that unsupervised pre-training helps because it acts like a regularizer.

A mathematical formalization of the regularization hypothesis is proposed by Erhan et al. (2009b): unsupervised pre-training is equivalent to adding an infinite penalty on

solutions that are outside of a particular region of parameter space. This regularizer corresponds to a data-dependent prior on parameters θ obtained through unsupervised learning.

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

For pre-trained models, the prior is

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

where the set of regions R_k partition a bounded region of parameter space and each region R_k of volume v_k corresponds to a basin of attraction of supervised gradient descent. The π_k are the prior probabilities that unsupervised pre-training will fall in region R_k. For the models without pre-training, there is also a prior with the same form, but with different probabilities r_k:

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

A more intuitive way to think of pre-training as a regularizer is by considering a particular initialization point as *implicitly* imposing constraints on the parameters of the network. These constraints specify which minimum (out of the large number of possible local minima) of the objective function is desired.

The optimization effect/hypothesis and the regularization effect/hypothesis seem at odds but are not necessarily incompatible, and one of the objectives of this paper is to help understand the subtle interplay between their effects.

Under the regularization hypothesis, its effect is very different from ordinary regularization due to smoothing (such as L_2 or L_1 regularization), and instead relies on capturing possibly very complicated structure in the input distribution $P(X)$, with the assumption that the true target conditional distribution $P(Y|X)$ and $P(X)$ share structure as functions of X (Bengio, 2009).

As stated in the introduction, the results in this paper support a regularization explanation. We should stress that the pre-training effect is unusual among regularizers and to simply state that pre-training is a regularizer is to undermine somewhat the significance of its effectiveness. The core of it is the restriction imposed by the unsupervised pre-training phase on the regions of parameter space that stochastic gradient descent can explore during the supervised phase. These regions correspond to parameters that are good at modeling $P(X)$; as with other semi-supervised methods, the effectiveness of the pre-training strategy for initialization will be limited to how much $P(X)$ is helpful in learning $P(Y|X)$.

4 Data and Setup

Two datasets were used to perform the experiments. The first is the `MNIST` digit classification dataset by LeCun et al. (1998), containing 60,000 training and 10,000 testing examples of 28x28 handwritten digits. The second is the `InfiniteMNIST` 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.

The models used are Deep Belief Networks containing Bernoulli RBM layers trained with standard Contrastive Divergence (Hinton et al., 2006), Stacked Denoising Auto-Encoders (Vincent et al., 2008), and standard feed-forward multi-layer neural networks, each with 1–3 hidden layers. Each hidden layer contains the same number of hidden units, which is a hyperparameter (the optimal number is usually in the 800-1200 units/layer range). The other hyperparameters are the unsupervised and supervised learning rates 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. The standard feed-forward networks are trained using 10 million supervised updates, with the Negative Log-Likelihood (NLL) of the correct class as the training objective. 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. Unsupervised learning is performed in a greedy layer-wise fashion.

5 Expanding Previous Results

The results of Erhan et al. (2009b) seem to point to a regularization explanation for the behavior of pre-training. These experiments were performed on Stacked Denoising Auto-Encoders (SDAE): we wanted to verify that similar conclusions could be drawn from experiments with other deep architectures, notably Deep Belief Networks.

We extend the observation made by Erhan et al. (2009b) regarding the influence of the size of the hidden layer on the generalization effect of pre-training. Figure 1 shows that DBNs behave qualitatively like SDAEs, in the sense that pre-training architectures with smaller layers hurts performance. Such results seems to re-verify the observation 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 reg-

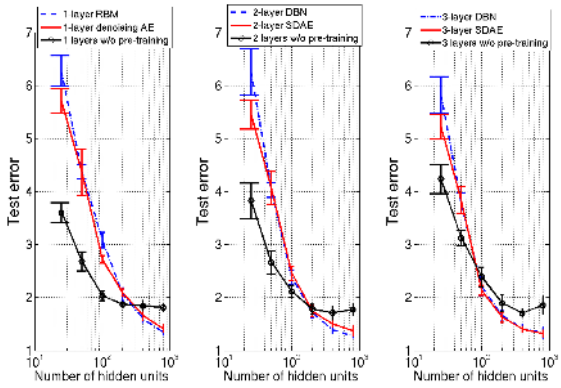

Figure 1: Effect of layer size and pre-training on DBNs, 1–3 hidden layer networks trained on MNIST.

ularization, generalization is hurt, and it is hurt more with unsupervised pre-training presumably because of the extra regularization effect.

Erhan et al. (2009b) made a claim that supervised gradients behave differently depending on the depth of the layer. The authors performed a limited experiment in which they studied the effect of selectively training layers in a 2-layer network. In this paper, we expand on those results; in Figure 2 (top) we explore the link between pre-training and the depth of the layer in more detail. The setup is as follows: 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).

The results are ambiguous w.r.t. the claim 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. Note how the final training error (after the same number of epochs) becomes *worse* with pre-training of more layers. This is consistent with a regularization interpretation of the effect of pre-training.

As mentioned in Section 2, Bengio et al. (2007) performed an experiment in which they constrained the size of the top hidden layer. They observed that doing so made it possible for the pre-trained networks to reach a better local minimum, compared to the networks without pre-training. The authors concluded that pre-trainining is an aid to optimization in deep architectures. However, early stopping on the validation set was performed, which means that networks were not allowed to train until a local minimum of the training criterion. We redo that experiment without early stopping, i.e., allowing the optimization to proceed.

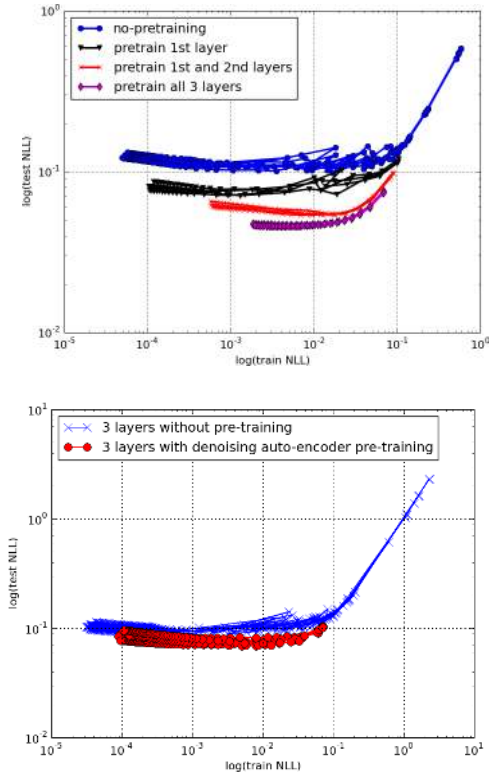

Figure 2: *Top*: MNIST, a plot of the log(train Negative Log-Likelihood) (NLL) vs. log(test NLL) at each epoch of training. We pre-train the first layer, the first two layers and all three layers using RBMs and randomly initialize the other layers; we also compare with the network whose layers are all randomly initialized. *Bottom*: 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.

What happens when we restrict the top layer to 20 units and measure the training error at convergence? Figure 2 shows that the training error is still higher for pre-trained networks even though the generalization error is lower[1]. This result now favors a regularization interpretation. What may happen is that early stopping prevented the networks without pre-training from moving too much towards their local minimum.

6 The Online Setting

An intuitive prediction from the perspective that pre-training is a regularizer is that its influence should diminish as the number of training examples increases. To test this and the claim that the basin of attraction induced by the

[1] If we compare Figure 2 (top) with 2 (bottom) we can also see that constraining the top layer of a pre-trained network allows it to reach a better training error

early examples is important even in a large-scale setting, we propose to study the online learning case, where the number of training examples is very large (potentially infinite). In a standard interpretation of a canonical (L_1/L_2 regularizer), its effect **diminishes** in such a setting. This is because the prior defined by it should be in principle be overcome by the likelihood from the ever-increasing data.

In this paper we discuss the surprising result that the effect of pre-training is **maintained** as the size of the dataset grows. We shall argue that the interplay between the non-convexity of the training objective and the clever unsupervised initialization technique is the reason for this.

6.1 `InfiniteMNIST`

The next set of results point in this direction and is the most surprising finding of this paper. Figure 3 shows the online classification error (on the next block of examples) for 6 architectures 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* means that we are optimizing well the training criterion. We can draw several observations from these results. First, 3-layer networks without pre-training are worse at generalization, compared to the 1-layer equivalent. It seems that even in an online setting, with very large amounts of data, optimizing deep networks is harder than shallow ones. Second, 3-layer SDAE models seem to generalize better than 3-layer DBNs. Finally and most surprisingly, the pre-training advantage **does not vanish** as the number of training examples increases, on the contrary. These results seem to support an optimization effect explanation for pre-training.

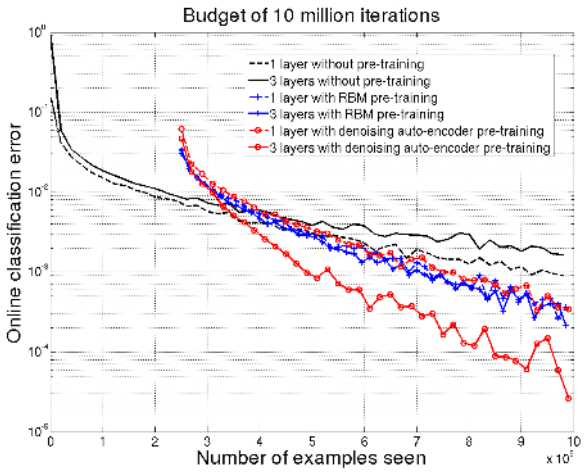

Figure 3: Comparison between 1 and 3-layer networks trained on `InfiniteMNIST`. 3-layer models have 800-1200 units/layer, 1-layer models have 2500 units in the hidden layer.

Note that the number of hidden units of each model is a hyperparameter. So theoretical results, such as the Universal Approximation Theorem, suggest that 1-layer networks without pre-training should in principle be able to represent the input distribution as capacity and data grow, as is the case in this experiment[2]. Instead, without pre-training, it seems that the networks are not able to take advantage of the additional capacity, which again points towards an 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 and favors a regularization interpretation.

6.2 The Influence of 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 4 shows the outcome of such an analysis. The samples at the beginning[3] 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 significantly *lower* than the variance of the supervised network. Such results imply that unsupervised pre-training can be seen as a sort of **variance reduction technique**, consistent with a regularization role. Finally, both networks have higher output variances as a function of the *last examples* used for optimization.

[2] In a limited sense, of course, since we are obviously not able to explore unbounded layer sizes and datasets.

[3] Which are *unsupervised* examples, for the red curve.

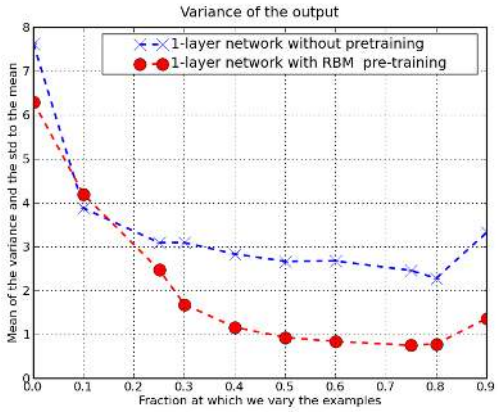

Figure 4: Variance of the output of a trained network with 1 layer. The variance (y-axis) is computed as a function of the point at which we vary the training samples (x-axis). Note that the 0.25 mark corresponds to the start of pre-training (for the red curve).

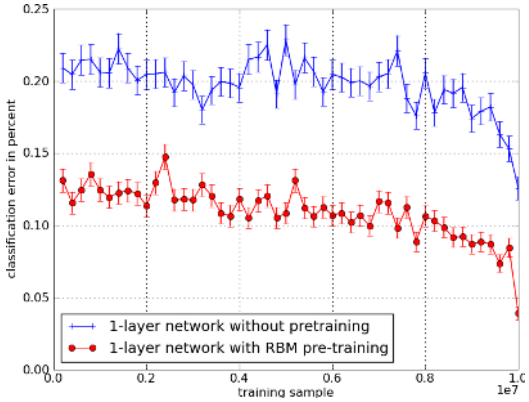

Figure 5: Error of 1-layer network with RBM pre-training and without, on the 10 million examples from `InfiniteMNIST`, used for training it. The errors are calculated in the same order as the examples were presented during training.

6.3 Evaluating The Training Error

Another experiment that shows the effects of large-scale online stochastic non-convex optimization is in the setting of `InfiniteMNIST`, where we compute the error on the *training set*, in the same order that we presented the examples to the models. Figure 5 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. Second, we observe that the pre-trained model is better across the board *on the training set*. This fits well with an optimization explanation. Note how this differs from what was seen with `MNIST`, where the training error was larger with pre-training (Erhan et al., 2009b). Finally, note that even the most recently seen examples, the pre-trained model (which is regularized, according to our hypothesis) performs better than the model without pre-training. On the surface, one would expect a regularized model to perform *worse* on the most recently seen examples because of additional constraints (via regularization), compared to the unregularized model. In reality, because of the non-convexity of the training criterion, both models (with and without pre-training) are constrained by the starting point of the supervised optimization; and the starting point of the pre-trained model is the basin of the attraction that is defined by unsupervised learning, which seems to be provide better generalization performance.

6.4 Filter visualization

Figure 6 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. (2009a)[4]. 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). Different layers change differently: the first layer changes least, while supervised training has more effect on the 3rd layer.

Figure 6 also shows the filters learned by a standard 3-layer network on the same data. The filters seem less interpretable, compared to the ones learned by the DBN, but more importantly they seem qualitatively very different from the ones learned by the DBN, even after the supervised phase. Such an observation is in line with the findings of Erhan et al. (2009b), whose function space visualizations of networks with and without pre-training showed that they explore essentially different regions of the function space; the filter visualizations reconfirm this in the qualitative parameter space.

These results support the claim regarding the difficulty of training deeper layers, but, more crucially, they demonstrate that the early dynamics of stochastic gradient descent can make training "stuck" in a region of the parameter space that is essentially not accessible for networks that are trained in a purely supervised way.

6.5 Selective Pre-training and The Effect of Canonical Regularizers

In Figure 7 we explore the link between pre-training and the depth of the layer for `InfiniteMNIST`. The setup is

[4]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 minimum of the activation function, starting from a random initialization point.

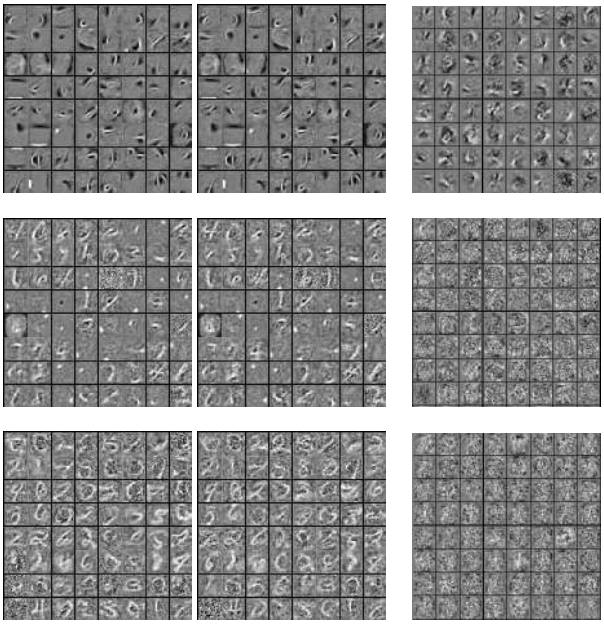

Figure 6: Visualization of a selection of filters learned by a DBN (left and middle) and by a standard 3-layer network trained on `InfiniteMNIST`. Both networks have 1000 units per layer. The left figures contain a visualization of DBN filters after pre-training, the middle ones picture the same units after supervised fine-tuning, while right ones picture units from the purely supervised network. From top to bottom: units from the 1st, 2nd and 3rd layers, respectively.

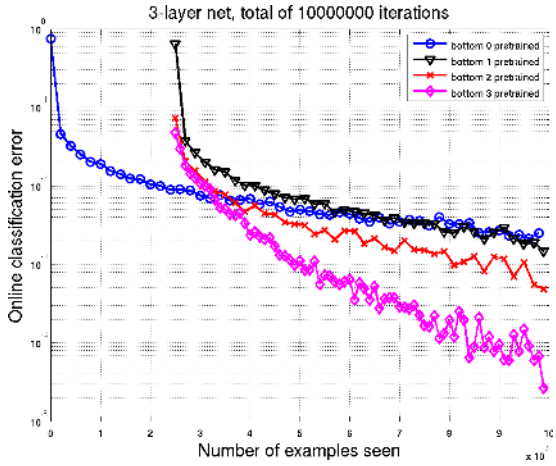

Figure 7: `InfiniteMNIST`, the online classification error. We pre-train the first layer, the first two layers or all three layers using denoising auto-encoders and leave the rest of the network randomly initialized.

the similar to the one in Figure 2, except this time we show the online error. The results are comparable to the ones in Section 2.

Finally, an alternative hypothesis to the regularization explanation 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 of a network without pre-training and found that while in some cases a small penalty could in principle help, the gain is nowhere near as large as it is with pre-training, and thus such a hypothesis can be rejected.

7 Discussion

Our findings support the idea that small perturbations in the trajectory followed by stochastic gradient descent (SGD) in the parameter space induce greater changes early on. This is due to the fact that early in training, weight changes tend to increase their magnitude and to add nonlinearity to the network. As training continues, the set of regions that are accessible by SGD is becoming smaller and smaller. This means that training is "trapped" in the *basin of attraction* that is defined by the early perturbations of the parameter space trajectory, and it is harder for SGD to "escape" from such a basin[5].

This implies that even in the presence of a very large (effectively infinite) amount of supervised data, SGD can suffer from a degree of *over-fitting* to the training data that is presented early on. This means that one can see pre-training as a (clever) way of interacting with the optimization process, by defining the starting point of the supervised training process and "trapping" it in the basin of attraction corresponding to parameters that are useful for performing unsupervised learning. As our results show, the effect of pre-training is indeed persistent even in the large-scale case.

This hypothesis states that due to the non-convexity of the supervised training criterion, early examples have a disproportionate influence on the outcome of the training procedure. This has ramifications in many of our results and is the reason why pre-training is still effective even in a large-scale setting where it is used only for initialization, where supervised training does not seem to escape from the basin of attraction defined by the initialization, as we add more examples. Thus, in contrast with classical regularizers, the effect of pre-training *does not disappear*.

With a small training set, one is usually not particularly interested in minimizing the training error on it, because over-fitting is typically an issue and because the training error is not a good way to separate the performance of two models. In such a setting, pre-training helps find better minima in terms of generalization performance. However, in a large-scale setting, as seen in Figure 5, *finding a better local minimum will matter* and better (stronger) optimiza-

[5] This would behave on the outside like the "critical period" phenomena observed in neuroscience and psychology (Bornstein, 1987).

tion techniques should have a significant impact on generalization in such scenarios.

Future work should clarify our finding about the influence of early examples. If this is corroborated by future results, it may mean that we need to consider algorithms that reduce the effect of early examples, especially in a large-scale setting. Bengio et al. (2009)'s work on curriculum learning, whereby an easy-to-hard ordering of training examples seems to make a difference is an interesting connection to this work and should be further investigated.

Other work includes the analysis of semi-supervised techniques for training deep architectures where there is only one phase of training, such as work by Larochelle and Bengio (2008) and Weston et al. (2008). We would also like to understand how learning a generative model of $P(Y, X)$ has a regularization effect relative to just learning $P(Y|X)$, as shown by Ng and Jordan (2002)[6]. Finally, it would be helpful to investigate the influence of "pre-training" using the so-called Fisher kernels (Jaakkola & Haussler, 1999), and in which sense do the results that we present hold in that setting as well.

Understanding and improving deep architectures continues to be a challenge. We believe that in order to devise improved strategies for training deep architectures, one needs a better understanding of the mechanisms and difficulties that we face with them. We put forward a hypothesis that explains the underpinnings on pre-training and this hypothesis is well supported by extensive simulations.

Acknowledgements

This research was supported by funding from NSERC, MITACS, FQRNT, and the Canada Research Chairs. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

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

Bengio, Y., Delalleau, O., & Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf and J. Platt (Eds.), *Nips 18*, 107–114. Cambridge, MA: MIT Press.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *NIPS 19* (pp. 153–160). MIT Press.

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

[6]Note that in that case, the difference in performance diminishes as the training set size increases.

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

Bornstein, M. H. (1987). *Sensitive periods in development : interdisciplinary perspectives / edited by marc h. bornstein*. Lawrence Erlbaum Associates, Hillsdale, N.J. :.

Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009a). *Visualizing higher-layer features of a deep network* (Technical Report 1341). Université de Montréal.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009b). The difficulty of training deep architectures and the effect of unsupervised pre-training. *AISTATS'2009* (pp. 153–160).

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

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

Jaakkola, T. S., & Haussler, D. (1999). Exploiting generative models in discriminative classifiers. *NIPS 11* (pp. 487–493). MIT Press, Cambridge, MA.

Larochelle, H., & Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines (pp. 536–543.). Helsinki, Finland.

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

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation (pp. 473–480.). Corvallis, OR.

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

Loosli, G., Canu, S., & Bottou, L. (2007). Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste and J. Weston (Eds.), *Large scale kernel machines*, 301–320. Cambridge, MA.: MIT Press.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *NIPS 14* (pp. 841–848).

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. *NIPS 19* (pp. 1137–1144). MIT Press.

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

Weston, J., Ratle, F., & Collobert, R. (2008). Deep learning via semi-supervised embedding. *Proc. ICML 2008* (pp. 1168–1175). New York, NY, USA.

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