Greedy Layer-Wise Training of Deep Networks

Yoshua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle
Dept. IRO, Université de Montréal
P.O. Box 6128, Downtown Branch, Montreal, H3C 3J7, QC, Canada
{bengioy,lamblinp,popovicd,larocheh}@iro.umontreal.ca

Technical Report 1282
Département d'Informatique et Recherche Opérationnelle

August 21st, 2006

Abstract

Deep multi-layer neural networks have many levels of non-linearities, which allows them to potentially represent very compactly highly non-linear and highly-varying functions. However, until recently it was not clear how to train such deep networks, since gradient-based optimization starting from random initialization appears to often get stuck in poor solutions. Hinton et al. recently introduced a greedy layer-wise unsupervised learning algorithm for Deep Belief Networks (DBN), a generative model with many layers of hidden causal variables. In the context of the above optimization problem, we study this algorithm empirically and explore variants to better understand its success and extend it to cases where the inputs are continuous or where the structure of the input distribution is not revealing enough about the variable to be predicted in a supervised task.

1 Introduction

Recent theoretical analyses (Bengio, Delalleau, & Le Roux, 2006) of modern non-parametric machine learning algorithm such as kernel machines and graph-based manifold and semi-supervised learning algorithms suggest fundamental limitations of some learning algorithms. The problem is clear in kernel-based approaches when the kernel is *"local"* (e.g. the Gaussian kernel), i.e. $K(x, y)$ converges to a constant when $||x - y||$ increases. These analyses point to the difficulty of learning *"highly-varying functions"*, i.e. functions that have a large number of *"variations"* in the domain of interest, e.g., they would require a large number of pieces to be well represented by a piecewise-linear approximation. Since the number of pieces can be made to grow exponentially with the number of input variables, this problem is directly connected with the well-known curse of dimensionality for classical non-parametric learning algorithms (for regression, classification and density estimation). If the shapes of all these pieces are unrelated, one needs enough examples for each piece in order to generalize properly. However, if these shapes are related and can be predicted from each other, *"non-local" learning algorithms* have the potential to generalize to pieces not covered by the training set. Such ability would seem necessary for learning in complex domains such as Artificial Intelligence tasks (e.g., related to vision, language, speech, robotics, etc.).

One way to represent a highly-varying function compactly (with few parameters) is through the composition of many non-linearities. For example, the parity function with d inputs requires $O(2^d)$ examples

and parameters to be represented by a Gaussian SVM (Bengio et al., 2006), $O(d^2)$ parameters for a one-hidden-layer neural network, $O(d)$ parameters for a multi-layer network with d layers, and $O(1)$ parameters with a recurrent neural network. More generally, boolean functions expressible by d layers of combinatorial logic with $O(d)$ elements in each layer may require $O(2^d)$ elements when expressed with only 2 layers (Utgoff & Stracuzzi, 2002). Many more arguments can be brought to bare to strongly suggest that learning of more abstract functions is much more efficient when it is done sequentially, composing previously learned concepts in order to represent and learn more abstract concepts (Utgoff & Stracuzzi, 2002). When the representation of a concept requires an exponential number of elements, e.g. with a shallow circuit, the number of training examples required to learn the concept may also be impractical.

However, until recently, it was believed to be too difficult to train deep multi-layer neural networks (or other types of deep multi-layered trainable architectures). Empirically, deep networks were generally found to be not better, and often worse, than neural networks with one or two hidden layers (Tesauro, 1992).As this is a negative result, it has not been much reported in the machine learning literature. A reasonable explanation is that gradient-based optimization starting from random initialization appears to often get stuck in poor solutions for such deep networks.

Hinton, Osindero, and Teh (2006) recently introduced a greedy layer-wise unsupervised learning algorithm for Deep Belief Networks (DBN), a generative model with many layers of hidden causal variables. The training strategy for such networks may hold promise as a principle to solve the problem of training deep networks. Upper layers of a DBN are supposed to represent more "abstract" concepts that explain the input observation x, whereas lower layers extract "low-level features" from x. They learn simpler concepts first, and build on them to learn more abstract concepts. This strategy has not yet been much exploited in machine learning, but it is at the basis of the greedy layer-wise constructive learning algorithm for DBNs. In this paper we also study other applications of the greedy layer-wise constructive strategy, with auto-encoders and greedy layer-wise supervised learning, in order to get a better understanding of the possible advantages of this strategy. We hypothesize that three aspects of the learning algorithm for DBNs are particularly important: pre-training one layer at a time in a greedy way, using unsupervised learning at each layer in order to preserve information from the input, and fine-tuning the whole network with respect to the ultimate criterion of interest. However, as we found out, when DBNs are used as initialization for a supervised task, it may be advantageous to use information about the target in the greedy pre-training, at least for the first layer.

We first extend DBNs and their component layers, Restricted Boltzmann Machines (RBM), so that they can more naturally handle continuous values in input. Second, we perform experiments to better understand the advantage brought by the greedy unsupervised learning. The basic question to answer is whether or not this approach helps to solve a difficult optimization problem. Finally, we discuss a problem that occurs with the layer-wise greedy unsupervised procedure when the input distribution is not revealing enough of the conditional distribution of the target variable y given the input variable x. We evaluate a simple and successful solution to this problem.

2 Deep Belief Nets

Let $\mathbf{g}^i$ represent the hidden variables at layer i. The model is parametrized as follows:

$$P(x, \mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^\ell) = P(x|\mathbf{g}^1)P(\mathbf{g}^1|\mathbf{g}^2)\cdots P(\mathbf{g}^{\ell-2}|\mathbf{g}^{\ell-1})P(\mathbf{g}^{\ell-1}, \mathbf{g}^\ell)$$

where all the conditional layers $P(\mathbf{g}^i|\mathbf{g}^{i+1})$ are factorized conditional distribution for which computation of probability and sampling are very easy. In Hinton et al. (2006) one considers the hidden layer $\mathbf{g}^i$ a

binary random vector with elements $\mathbf{g}^i_j$:

$$P(\mathbf{g}^i|\mathbf{g}^{i+1}) = \prod_{j=1}^{n^i} P(\mathbf{g}^i_j|\mathbf{g}^{i+1}) \text{ with } P(\mathbf{g}^i_j = 1|\mathbf{g}^{i+1}) = \text{sigm}(-b^i_j - \sum_{k=1}^{n^{i+1}} W^i_{kj}\mathbf{g}^{i+1}_k) \quad (1)$$

where $\text{sigm}(x) = 1/(1+e^{-x})$, the b^i_j are called the *biases* (for unit j of layer i) and W^i is called the *weight matrix* for layer i. If we denote $\mathbf{g}^0 = x$, the generative model for the first layer $P(x|\mathbf{g}^1)$ also follows equation 1.

2.1 Restricted Boltzmann machines

The top-level prior $P(\mathbf{g}^{\ell-1}, \mathbf{g}^\ell)$ is a **Restricted Boltzmann Machine** (RBM) between layer $\ell - 1$ and layer ℓ. To lighten notation, consider a generic RBM with input layer activations $\mathbf{v}$ (for *visible* units) and hidden layer activations $\mathbf{h}$ (for *hidden* units). It has the following joint distribution:

$$P(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-\mathbf{h}'W\mathbf{v} - b'\mathbf{v} - c'\mathbf{h}} \quad (2)$$

where Z is the normalization constant for this distribution, b is the vector of biases for visible units, c is the vector of biases for the hidden units, and W is the weight matrix for the layer. Minus the argument of the exponential is called the **energy function**,

$$\text{energy}(\mathbf{v}, \mathbf{h}) = \mathbf{h}'W\mathbf{v} + b'\mathbf{v} + c'\mathbf{h}. \quad (3)$$

We denote the RBM parameters together with $\theta = (W, b, c)$. We denote $Q(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ the layer-to-layer conditional distributions associated with the above RBM joint distribution. The layer-to-layer conditionals associated with the RBM factorize and give rise to $P(\mathbf{v}_k = 1|\mathbf{h}) = \text{sigm}(-b_k - \sum_j W_{jk}\mathbf{h}_j)$ and $Q(\mathbf{h}_j = 1|\mathbf{v}) = \text{sigm}(-c_j - \sum_k W_{jk}\mathbf{v}_j)$.

2.2 Gibbs Markov chain and log-likelihood gradient in an RBM

Gibbs sampling from an RBM proceeds by sampling $\mathbf{h}$ given $\mathbf{v}$, then $\mathbf{v}$ given $\mathbf{h}$, etc. Denote $\mathbf{v}_t$ for the t-th $\mathbf{v}$ sample from that chain, starting at $t = 0$ with $\mathbf{v}_0$, the "input observation" for the RBM. Therefore, $(\mathbf{v}_k, \mathbf{h}_k)$ for $k \to \infty$ is a sample from the joint $P(\mathbf{v}, \mathbf{h})$. The log-likelihood of a value $\mathbf{v}_0$ under the model of the RBM is

$$\log P(\mathbf{v}_0) = \log \sum_{\mathbf{h}} P(\mathbf{v}_0, \mathbf{h}) = \log \sum_{\mathbf{h}} e^{-\text{energy}(\mathbf{v}_0, \mathbf{h})} - \log \sum_{\mathbf{v},\mathbf{h}} e^{-\text{energy}(\mathbf{v}, \mathbf{h})}$$

and its gradient with respect to $\theta = (W, b, c)$ is

$$\frac{\partial \log P(\mathbf{v}_0)}{\partial \theta} = -\sum_{\mathbf{h}_0} Q(\mathbf{h}_0|\mathbf{v}_0) \frac{\partial \text{energy}(\mathbf{v}_0, \mathbf{h}_0)}{\partial \theta} + \sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v}_k, \mathbf{h}_k) \frac{\partial \text{energy}(\mathbf{v}_k, \mathbf{h}_k)}{\partial \theta}$$

for $k \to \infty$. An unbiased sample is

$$-\frac{\partial \text{energy}(\mathbf{v}_0, \mathbf{h}_0)}{\partial \theta} + \frac{\partial \text{energy}(\mathbf{v}_k, \mathbf{h}_k)}{\partial \theta},$$

where $\mathbf{h}_0$ is a sample from $Q(\mathbf{h}_0|\mathbf{v}_0)$ and $(\mathbf{v}_k, \mathbf{h}_k)$ is a sample of the Markov chain. The idea of the Contrastive Divergence (Hinton, 2002) algorithm is to take k small (typically $k = 1$), i.e. run the chain for only one step.

A pseudo-code for Contrastive Divergence training (with $k = 1$) of an RBM with binomial input and hidden units is presented in the Appendix (Algorithm `RBMupdate`(x, ϵ, W, b, c)). This procedure is called repeatedly with x sampled from the training distribution for the RBM. To decide when to stop one may use a proxy for the training criterion, such as the reconstruction error $-\log P(\mathbf{v}_1 = x|\mathbf{v}_0 = x)$, whose average tends to flatten as training of the RBM progresses.

2.3 Greedy layer-wise training of a DBN

A greedy layer-wise training algorithm was proposed (Hinton et al., 2006) to train a DBN one layer at a time. We first train an RBM that takes the empirical data as input and models it. Denote $Q(\mathbf{g}^1|\mathbf{g}^0)$ the posterior over $\mathbf{g}^1$ associated with that trained RBM (we recall that $\mathbf{g}^0 = x$ with x the empirical data). This gives rise to an "empirical" distribution $\widehat{p}^1$ over the first layer $\mathbf{g}^1$, when $\mathbf{g}^0$ is sampled from the data empirical distribution $\widehat{p}$: we have $\widehat{p}^1(\mathbf{g}^1) = \sum_{\mathbf{g}^0} \widehat{p}(\mathbf{g}^0) Q(\mathbf{g}^1|\mathbf{g}^0)$.

The basic idea of the greedy layer-wise training algorithm for DBNs is that after training the top-level RBM of a ℓ-level DBN, one can change the interpretation of the RBM parameters to insert them in a $(\ell + 1)$-level DBN: the distribution $P(\mathbf{g}^{\ell-1}|\mathbf{g}^\ell)$ from the RBM associated with layers $\ell - 1$ and ℓ is kept as part of the DBN generative model. On the other hand, the RBM's conditional in the other direction, denoted $Q(\mathbf{g}^\ell|\mathbf{g}^{\ell-1})$, is used as an approximation of the DBN's conditional $P(\mathbf{g}^\ell|\mathbf{g}^{\ell-1})$. After this operation, the DBN marginal distribution $P(\mathbf{g}^\ell)$ will thus be the one induced by the upper layers of the DBN (initially the RBM associated with layers ℓ and $\ell + 1$) and not by the marginal $Q(\mathbf{g}^\ell)$ induced by the RBM associated with layers ℓ and $\ell - 1$. The samples of $\mathbf{g}^{\ell-1}$, with empirical distribution $\widehat{p}^{\ell-1}$, can be converted stochastically in samples of $\mathbf{g}^\ell$ by $\widehat{p}^\ell(\mathbf{g}^\ell) = \sum_{\mathbf{g}^{\ell-1}} \widehat{p}^{\ell-1}(\mathbf{g}^{\ell-1}) Q(\mathbf{g}^\ell|\mathbf{g}^{\ell-1})$. Although $\widehat{p}^\ell$ cannot be represented explicitly it is very easy to sample unbiasedly from it (pick an input example from $\widehat{p}$ and propagate it stochastically through the $Q(\mathbf{g}^i|\mathbf{g}^{i-1})$ at each level). As a very nice side benefit, one obtains an approximation of the posterior for all the hidden variables in the DBN, at all levels, given an input $\mathbf{g}^0 = x$.

Note that if we consider all the layers of a DBN from level i to the top, we have a smaller DBN, which generates the marginal distribution $P(\mathbf{g}^i)$ for the complete DBN. The motivation for the greedy procedure is that a partial DBN with $\ell - i$ levels starting above level i may provide a better model for $P(\mathbf{g}^i)$ than does the RBM initially associated with level i itself.

The above greedy procedure is justified using a variational bound (Hinton et al., 2006). As a consequence of that bound, when inserting an additional layer, if it is initialized appropriately and has enough units, one can guarantee that initial improvements on the training criterion for the next layer (fitting $\widehat{p}^\ell$) will yield improvement on the training criterion for the previous layer (likelihood with respect to $\widehat{p}^{\ell-1}$).

The greedy layer-wise training algorithm for DBNs is quite simple, as illustrated by the pseudo-code in Algorithm `TrainUnsupervisedDBN` of the Appendix.

2.4 Supervised fine-tuning

As a last training stage, it is possible to fine-tune the parameters of all the layers together. For example Hinton et al. (2006) propose to use the wake-sleep algorithm (Hinton, Dayan, Frey, & Neal, 1995) to

continue unsupervised training. Hinton et al. (2006) also propose to optionally use a mean-field approximation of the posteriors $P(\mathbf{g}^i|\mathbf{g}^0)$, by replacing the samples $\mathbf{g}_j^{i-1}$ at level $i-1$ by their bit-wise mean-field expected value μ_j^{i-1}, with $\mu^i = \text{sigm}(-b^i - W^i\mu^{i-1})$. According to these propagation rules, the whole network now deterministically computes internal representations as functions of the network input $\mathbf{g}^0$. After unsupervised pre-training of the layers of a DBN following Algorithm `TrainUnsupervisedDBN` (see Appendix) the whole network can be further optimized by gradient descent with respect to any deterministically computable training criterion that depends on these representations. For example, this can be used (Hinton 2006, unpublished) to fine-tune a very deep auto-encoder, minimizing a reconstruction error. It is also possible to use this as initialization of all except the last layer of a traditional multi-layer neural network, using gradient descent to fine-tune the whole network with respect to a supervised training criterion.

Pseudo-code for supervised fine-tuning is given in the Appendix (Algorithm `DBNSupervisedFineTuning`), as part of the global supervised learning algorithm `TrainSupervisedDBN` (both in the Appendix). Note that better results were obtained when using a *twenty-fold larger learning rate* for gradient descent on the supervised criterion (mean squared error or cross-entropy, in our experiments) than in the contrastive divergence update.

3 Extension to continuous-valued inputs

With the binary units introduced for RBMs and DBNs in Hinton et al. (2006) one can "cheat" and handle continuous-valued inputs by scaling them to the (0,1) interval and considering each input continuous value as the probability for a binary random variable to take the value 1. This has worked well for pixel gray levels, but it may be inappropriate for other kinds of input variables. Previous work on continuous-valued input in RBMs include (Chen & Murray, 2003), in which noise is added to sigmoidal units, and the RBM forms a special form of Diffusion Network (Movellan, Mineiro, & Williams, 2002). We concentrate here on simple extensions of the RBM framework in which only the energy function and the allowed range of values are changed.

Linear energy: exponential or truncated exponential

Consider a unit u of an RBM, connected to units $\mathbf{z}$. $p(u|\mathbf{z})$ can be obtained from the terms of the energy function that contain u, which can be grouped in $ua(\mathbf{z})$ for linear energy functions as in eq. (3), where $a(\mathbf{z}) = b + w'\mathbf{z}$ with b the bias of unit u and w the vector of weights connecting unit u to units $\mathbf{z}$. If we allow u to be anywhere in some interval I, the conditional density of u becomes $p(u|\mathbf{z}) = \frac{exp(-ua(\mathbf{z}))\mathbf{1}_{u\in I}}{\int_v exp(-va(\mathbf{z}))\mathbf{1}_{v\in I}dv}$. When $I = [0, \infty)$, this is an exponential density with parameter $a(\mathbf{z})$, and the normalizing integral equals $-1/a(\mathbf{z})$. Computing the density, the expected value ($= -a(\mathbf{z})$), and sampling are all easy.

Alternatively, if I is a closed interval (as in many applications of interest), or if we would like to use such a unit as a hidden unit with *non-linear expected value*, the above density is a *truncated exponential*. For simplicity we consider the case $I = [0, 1]$ here, for which the normalizing integral is $\frac{1}{a(\mathbf{z})} - \frac{exp(-a(\mathbf{z}))}{a(\mathbf{z})}$. The conditional expectation of u given $\mathbf{z}$ is interesting because it has a sigmoidal-like saturating and monotone non-linearity: $E[u|\mathbf{z}] = \frac{1}{1-exp(a(\mathbf{z}))} + \frac{1}{a(\mathbf{z})}$. A sampling from the truncated exponential is easily obtained from a uniform sample U: $F^{-1}(U) = \frac{\log(1-U\times(1-exp(-a(\mathbf{z}))))}{a(\mathbf{z})}$.

In both truncated and not truncated cases, the Contrastive Divergence updates are the same as for binomial units, since they only depend on the derivative of the energy with respect to the parameters. Only sampling is changed, according to the unit's conditional density.

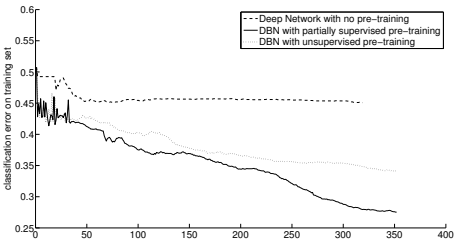

Figure 1: Evolution of training classification error on the Cotton price task, for deep network without pre-training, for DBN with unsupervised pre-training, and DBN with partially supervised pre-training. Illustrates the optimization difficulty of a deep network and the advantage of partially supervised training of the first layer.

Quadratic energy: Gaussian units

To obtain Gaussian-distributed units, one must add quadratic terms to the energy function. For a diagonal covariance matrix between units of the same layer, it is enough add terms of the form $\sum_i d_i^2 u_i^2$, where u_i is the continuous value of a Gaussian unit and d_i^2 is a positive parameter that is equal to the inverse of the variance of u_i. In this case only the mean will be conditional, and we obtain that for a unit u with inputs $\mathbf{z}$ and inverse variance parameter d^2, $E[u|\mathbf{z}] = -\frac{a(\mathbf{z})}{2d^2}$.

The Contrastive Divergence updates are obtained very easily by computing the derivative of the energy with respect to the parameters. For the linear parameters the derivatives are the same as for the binomial units. For quadratic parameter d, the derivative is simply $2du^2$.

Gaussian units were previously used, as hidden units of an RBM (with inputs binomial or multinomial) applied to an information retrieval task (Welling, Rosen-Zvi, & Hinton, 2005). Our interest here is to use them for continuous-valued inputs.

Using continuous-valued hidden units

Although we have introduced RBM units with continuous values to better deal with the representation of input variables, they could also be considered for use in the hidden layers, in replacement or complementing the binomial units which have been used in the past.

However, Gaussian and exponential hidden units have a weakness: the mean-field propagation through a Gaussian unit gives rise to a purely linear transformation. Hence if we have only such linear hidden units in a multi-layered network, the mean-field propagation function that maps inputs to internal representations would be completely linear. In addition, in a DBN containing only Gaussian units, one would only be able to model Gaussian data. On the other hand, combining Gaussian with other types of units could be interesting.

Instead, note that the conditional expectation of truncated exponential units is non-linear, and in fact involves a sigmoidal form of non-linearity applied to the weighted sum of its inputs.

Experiment 1

This experiment was performed on two data sets: the UCI repository Abalone data set (split in 2177 training examples, 1000 validation examples, 1000 test examples) and a financial data set. The latter has real-valued input variables representing averages of returns and squared returns for which the binomial approximation would seem inappropriate. The target variable is a function of Cotton futures contracts. There are 13 continuous input variables, that are averages of returns over different time-windows up to 504 days. There are 3135 training examples, 1000 validation examples, and 1000 test examples. The dataset is publicly available at `http://www.iro.umontreal.ca/~lisa/fin_data/`. We compare DBNs with Gaussian inputs and binomial hidden units with DBNs with binomial inputs. The networks have two hidden layers. All hyper-parameters are selected based on validation set performance.

	Abalone			Cotton		
	train.	valid.	test.	train.	valid.	test.
Deep Network with no pre-training	4.23	4.43	4.2	45.2%	42.9%	43.0%
Logistic regression	·	·	·	44.0%	42.6%	45.0%
DBN, binomial inputs, unsupervised	4.59	4.60	4.47	44.0%	42.6%	45.0%
DBN, binomial inputs, partially supervised	4.39	4.45	4.28	43.3%	41.1%	43.7%
DBN, Gaussian inputs, unsupervised	4.25	4.42	4.19	35.7%	34.9%	35.8%
DBN, Gaussian inputs, partially supervised	4.23	4.43	4.18	27.5%	28.4%	31.4%

Table 1: *Mean squared prediction error on Abalone task and classification error on Cotton task, showing improvement with Gaussian units.*

4 Understanding why the layer-wise strategy works

A reasonable explanation for the apparent success of the layer-wise training strategy for DBNs is that this unsupervised pre-training helps to mitigate the difficult optimization problem of deep networks by better initializing the weights of all the layers. In this section we discuss this hypothesis and present experiments that support it and clarify it.

Training each layer as an auto-encoder

We want to verify that the layer-wise greedy unsupervised pre-training principle can be applied to a more traditional setting for training a single layer of representation, i.e. with an auto-encoder. Each layer is trained to minimize a reconstruction cross-entropy. Let x the input vector with $x_i \in (0, 1)$. For a layer with weights matrix W, hidden biases column vector b and input bias column vector c, the reconstruction probability for bit i is $p_i(x)$, with the vector of probabilities $p(x) = \text{sigm}(-c - W\text{sigm}(b + W'x))$. The training criterion for the layer is the sum of negative log-likelihoods for predicting x from $p(x)$, for training examples x. For example if x is interpreted either as a sequence of bits or a sequence of bit probabilities, we used the reconstruction cross-entropy: $R = -\sum_i x_i \log p_i(x) + (1 - x_i) \log(1 - p_i(x))$. We report several experimental results using this training criterion for each layer, in comparison to the contrastive divergence algorithm for an RBM. Pseudo-code for a deep network obtained by training each layer as an auto-encoder is given in Appendix (Algorithm `TrainGreedyAutoEncodingDeepNet`).

One question that arises with auto-encoders in comparison with DBNs is whether the auto-encoders will fail to learn a useful representation when the number of units is not strictly decreasing from one layer to the next (since the networks could theoretically just learn to be the identity and perfectly minimize the reconstruction error). However, our experiments indicate that networks with non-decreasing sizes of hidden layers work well. This might be due to weight decay and stochastic gradient descent, preventing large weights: the optimization falls in a local minimum which corresponds to a meaningful transformation of the input (at least in the sense that it serves as a better initialization for supervised training of the whole network).

Greedy layer-wise supervised training

A reasonable question to ask is whether the fact that each layer is trained in an unsupervised way is critical or not. An alternative algorithm is supervised, greedy and layer-wise: train each new hidden layer as the hidden layer of a one-hidden layer supervised neural network (taking as input the output of the previously

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.3%	1.4%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.75%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

Table 2: *Classification error on MNIST training, validation, and test sets, with the best hyper-parameters according to validation error, with and without pre-training, using purely supervised or purely unsupervised pre-training.*

trained layers), and then throw away the output layer and use the trained hidden layer as pre-training initialization, to map the output of the previous layers to a hopefully better representation.

Pseudo-code for a deep network obtained by training each layer as the hidden layer of a supervised one-hidden-layer neural network is given in Appendix (Algorithm `TrainGreedySupervisedDeepNet`).

Experiment 2.

We compared the performance on the MNIST digit classification task obtained with five algorithms: (a) DBN, (b) deep network whose layers are trained as auto-encoders, (c) the supervised greedy layer-wise algorithm described above to pre-train each layer, (d) deep network (3 hidden layers) with no pre-training (random initialization), (e) shallow network (1 hidden layer) with no pre-training.

In all cases the final fine-tuning is done by adding a logistic regression layer on top of the network and training the whole network by stochastic gradient descent on the cross-entropy with respect to the target classification. In all cases the networks have the following architecture: 784 inputs, 10 outputs, 3 hidden layers with variable number of hidden units, chosen by trial and error and selected by validation set performance. The shallow network has a single hidden layer. In all cases an L2 weight decay is also optimized. The DBN was slower to train and less experiments were performed, so that longer training and more appropriately chosen sizes of layers and learning rates could yield better results (Hinton 2006, unpublished, reports an error rate of 1.15% on the MNIST test set).

On MNIST, differences of more than .1% are statistically significant. The results in Table 1 suggest that the auto-encoding criterion can yield performance comparable to the DBN when the layers are finally tuned in a supervised fashion. They also clearly show that the greedy unsupervised layer-wise pre-training gives much better results than the standard way to train a deep network (with no greedy pre-training) or a shallow network, and that, without pre-training, deep networks tend to perform worse than shallow networks. The results also suggest that *unsupervised greedy layer-wise pre-training can perform significantly better than purely supervised greedy layer-wise pre-training*. A possible explanation is that the greedy supervised procedure is **too greedy**: in the learned hidden units representation it may discard some of the information about y, information that cannot be captured easily by a one-hidden-layer neural network but could be captured by composing more hidden layers.

Experiment 3

However, there is something troubling in these results: all the networks, even those without greedy layer-wise pre-training, perform almost perfectly on the *training set*, which would appear to contradict the hypothesis that the main effect of the layer-wise greedy strategy is to help the optimization (with poor optimization one would expect poor training error). A possible explanation coherent with our initial hypothesis and with the above results is the following: without pre-training the lower layers are initialized poorly, but still allowing the top layer to learn the training set almost perfectly, because the output layer

	training error	validation error	test error
Deep net, auto-associator pre-training	0%	1.4%	1.6%
Deep net, supervised pre-training	0%	1.8%	1.9%
Deep net, no pre-training	.59%	2.1%	2.2%
Shallow net, no pre-training	3.6%	4.7%	5.0%

Table 3: *Classification error on MNIST with 20 hidden units on top layer.*

and the last hidden layer can be trained as easily as with a standard shallow neural network. To test that hypothesis, we performed a second series of experiments in which we constrain the top hidden layer to be small (20 hidden units).

The results in Table 2 clearly confirm the hypothesis previously introduced to explain the low training error for networks without pre-training. In addition, the results obtained without pre-training were found to have extremely large variance indicating very high sensitivity to initial conditions. Overall, the results in the two tables and in Figure 1 are consistent with the hypothesis that the greedy layer-wise procedure essentially helps to better optimize the deep networks, but they introduce some interesting subtleties in the understanding of what is happening.

Continuous training of all layers of a DBN

With the layer-wise training algorithm for DBNs (`TrainUnsupervisedDBN` in Appendix), one element that we would like to dispense with is having to decide the number of training iterations for each layer. When should we add a layer? There are many possible ways to address this question, but it would be good if we did not have to explicitly add layers one at a time, i.e., if we could train all layers simultaneously, but keeping the "greedy" idea that *each layer is pre-trained to model its input, ignoring the effect of higher layers*. To achieve this it is sufficient to insert a line in `TrainUnsupervisedDBN`, so that `RBMupdate` is called on all the layers and the stochastic hidden values are propagated all the way up. Experiments with this variant demonstrated that it works at least as well as the original algorithm. The advantage is that we can now have a single stopping criterion (for the whole network). There is a slight disadvantage in computation time, since we do more computations initially (on the upper layers), which might be wasted (before the lower layers converge to a decent representation), but time may be saved on optimizing hyper-parameters. Note that this variant may be more appealing for on-line training on very large data-sets, where one would never cycle back on the training data.

5 Dealing with uncooperative input distributions

In classification problems such as MNIST where classes are well separated, the structure of the input distribution $p(x)$ naturally contains much information about the target variable y. Imagine a supervised learning task in which the input distribution is mostly unrelated with y. In regression problems, which we are interested in studying here, this problem could be much more prevalent. For example imagine a task in which $x \sim p(x)$ and $y = f(x) + \text{noise}$ (e.g. p is Gaussian and $f = \text{sinus}$) with no particular relation between p and f. In such settings we cannot expect the unsupervised greedy layer-wise pre-training procedure to help in training deep networks. To deal with such uncooperative input distributions, we propose to train each layer with a mixed training criterion that combines the unsupervised objective (modeling or reconstructing the input) and a supervised objective (helping to predict the target). A very simple

algorithm thus adds the updates on the hidden layer weights from the unsupervised algorithm (Contrastive Divergence or reconstruction error gradient) with the updates from the gradient on a supervised prediction error, using a temporary output layer, as with the greedy layer-wise supervised training algorithm. In our experiments it appeared sufficient to perform that partial supervision with the **first layer only**, since once the predictive information about the target y is "forced" into the representation of the first layer, it tends to stay in the upper layers.

A pseudo-code for that partially supervised training algorithm for the first (or later layer) is given in Algorithm `TrainPartiallySupervisedLayer` (in the Appendix).

The results in table 1 clearly show the advantage of the partially supervised greedy training algorithm, in the case of the financial dataset.

6 Conclusion

In this paper we are mainly interested in Deep Belief Networks applied to supervised learning tasks. The three principal contributions of this paper are the following. First we extended RBMs and DBNs in new ways to naturally handle continuous-valued inputs, showing examples where much better predictive models can thus be obtained. Second, we performed experiments which support the hypothesis that the greedy layer-wise training strategy helps to optimize deep networks, but that it is important to have an unsupervised component to train each layer (a fully supervised greedy layer-wise strategy does not work as well). These experiments also suggest that each layer can be trained as an auto-associator instead of as an RBM, yielding comparable results. Finally, we studied the problems with the above strategy when the structure of the input distribution is not revealing enough of the conditional density of y given x, and we proposed a simple fix to the greedy strategy that can yield significant improvements for datasets where this problem occurs.

References

Bengio, Y., Delalleau, O., & Le Roux, N. (2006). The curse of highly variable functions for local kernel machines. In *Advances in Neural Information Processing Systems 18*. MIT Press.

Chen, H., & Murray, A. (2003). A continuous restricted boltzmann machine with an implementable training algorithm. *IEE Proceedings of Vision, Image and Signal Processing*, *150*(3), 153–158.

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

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*(8), 1771–1800.

Hinton, G., Dayan, P., Frey, B., & Neal, R. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, *268*, 1558–1161.

Movellan, J., Mineiro, P., & Williams, R. (2002). A monte-carlo EM approach for partially observable diffusion processes: theory and applications to neural networks. *Neural Computation*, *14*, 1501–1544.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277.

Utgoff, P., & Stracuzzi, D. (2002). Many-layered learning. *Neural Computation*, *14*, 2497–2539.

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

Appendix: Algorithms

Algorithm 1 `RBMupdate`$(\mathbf{v}_0, \epsilon, W, b, c)$

This is the RBM update procedure for binomial units. It also works for exponential and truncated exponential units, and for the linear parameters of a Gaussian unit (using the appropriate sampling procedure for Q and P). It can be readily adapted for the variance parameter of Gaussian units, as discussed in the text.

v_0 is a sample from the training distribution for the RBM

ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence

W is the RBM weight matrix, of dimension (number of hidden units, number of inputs)

b is the RBM biases vector for hidden units

c is the RBM biases vector for input units

---

for all hidden units i **do**
- compute $Q(\mathbf{h}_{0i} = 1|\mathbf{v}_0)$ (for binomial units, $\text{sigm}(-b_i - \sum_j W_{ij}\mathbf{v}_{0j})$)
- sample $\mathbf{h}_{0i}$ from $Q(\mathbf{h}_{0i} = 1|\mathbf{v}_0)$

end for

for all visible units j **do**
- compute $P(\mathbf{v}_{1j} = 1|\mathbf{h}_0)$ (for binomial units, $\text{sigm}(-c_j - \sum_i W_{ij}\mathbf{h}_{0i})$)
- sample $\mathbf{v}_{1j}$ from $P(\mathbf{v}_{1j} = 1|\mathbf{h}_0)$

end for

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i} = 1|\mathbf{v}_1)$ (for binomial units, $\text{sigm}(-b_i - \sum_j W_{ij}\mathbf{v}_{1j})$)

end for

- $W \leftarrow W - \epsilon(\mathbf{h}_0\mathbf{v}_0' - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1)\mathbf{v}_1')$
- $b \leftarrow b - \epsilon(\mathbf{h}_0 - Q(\mathbf{h}_{1\cdot} = 1|\mathbf{v}_1))$
- $c \leftarrow c - \epsilon(\mathbf{v}_0 - \mathbf{v}_1)$

Algorithm 2 `TrainUnsupervisedDBN`($\widehat{p}, \epsilon, L, n, W, b$)

Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM by contrastive divergence.

$\widehat{p}$ is the input training distribution for the network
ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence
L is the number of layers to train
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L

• initialize $b^0 = 0$
for $\ell = 1$ to L **do**
  • initialize $W^i = 0$, $b^i = 0$
  while not stopping criterion **do**
    • sample $\mathbf{g}^0 = x$ from $\widehat{p}$
    for $i = 1$ to $\ell - 1$ **do**
      • sample $\mathbf{g}^i$ from $Q(\mathbf{g}^i | \mathbf{g}^{i-1})$
    end for
    • `RBMupdate`($\mathbf{g}^{\ell-1}, \epsilon, W^\ell, b^\ell, b^{\ell-1}$)
  end while
end for

Algorithm 3 `PreTrainGreedyAutoEncodingDeepNet`$(\widehat{p}, C, \epsilon, L, n, W, b)$

Initialize all layers except the last in a multi-layer neural network, in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an auto-associator that tries to reconstruct its input.

$\widehat{p}$ is the training distribution for the network

$C = -\log P_\theta(u)$ is a reconstruction error criterion that takes θ and u as input, with θ the parameters of a predicted probability distribution and u an observed value.

ϵ is a learning rate for the stochastic gradient descent in reconstruction error

L is the number of layers to train

$n = (n^0, \ldots, n^L)$, with n^0 the inputs size and n^i the number of hidden units in each layer $i \geq 1$.

W^i is the weight matrix for level i, for i from 1 to L

b^i is the bias vector for level i, for i from 0 to L

- initialize $b^0 = 0$.
- define $\mu^0(x) = x$.

for $\ell = 1$ to L **do**

- initialize $b^\ell = 0$.
- initialize temporary parameter vector $c^\ell = 0$.
- initialize W^ℓ by sampling from uniform$(-a, a)$, with $a = 1/n^{\ell-1}$.
- define the ℓ-th hidden layer output $\mu^\ell(x) = \text{sigm}(b^\ell + W^\ell \mu^{\ell-1}(x))$.
- define the ℓ-th hidden layer reconstruction parameter function, e.g. in the binomial case $\theta^\ell = \text{sigm}(c^\ell + W^{\ell\prime} \mu^\ell(x))$ is the vector of probabilities for the each bit to take value 1.

while not stopping criterion **do**

for $i = 1$ to $\ell - 1$ **do**

- compute $\mu^i(x)$ from $\mu^{i-1}(x)$.

end for

- compute $\mu^\ell(x)$ from $\mu^{\ell-1}(x)$.
- compute reconstruction probability parameters θ^ℓ from $\mu^\ell(x)$.
- compute the error C in reconstructing $\mu^{\ell-1}$ from probability with parameters θ^ℓ.
- compute $\frac{\partial C}{\partial \omega}$, for $\omega = (W^\ell, b^\ell, c^\ell)$
- update layer parameters: $\omega \leftarrow \omega - \epsilon \frac{\partial C}{\partial \omega}$

end while

end for

Algorithm 4 `TrainSupervisedDBN`($\widehat{p}, C, \epsilon_{CD}, \epsilon_C, L$,$n$,$W$,$b$,$V$)
Train a DBN for a supervised learning task, by first performing pre-training of all layers (except the output weights V), followed by supervised fine-tuning to minimize a criterion C.
$\widehat{p}$ is the supervised training distribution for the DBN, with (input,target) samples (x, y)
C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$
ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence
ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C
L is the number of layers
$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer
W^i is the weight matrix for level i, for i from 1 to L
b^i is the bias vector for level i, for i from 0 to L V is a weight matrix for the supervised output layer of the network

---

- Let $\widehat{p}_x$ the marginal over the input part of $\widehat{p}$
- TrainUnsupervisedDBN($\widehat{p}_x, \epsilon_{CD}, L, n, W, b$)
- DBNSupervisedFineTuning($\widehat{p}, C, \epsilon_C, L$,$n$,$W$,$b$,$V$)

Algorithm 5 `DBNSupervisedFineTuning`($\widehat{p}, C, \epsilon_C, L, n, W, b, V$)

After a DBN has been initialized by pre-training, this procedure will optimize all the parameters with respect to the supervised criterion C, using stochastic gradient descent.

$\widehat{p}$ is the supervised training distribution for the DBN, with (input,target) samples (x, y)

C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$

ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence

ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C

L is the number of layers

$n = (n^1, \ldots, n^L)$ is the number of hidden units in each layer

W^i is the weight matrix for level i, for i from 1 to L

b^i is the bias vector for level i, for i from 0 to L V is a weight matrix for the supervised output layer of the network

---

- Recursively define mean-field propagation $\mu^i(x) = E[\mathbf{g}^i | \mathbf{g}^{i-1} = \mu^{i-1}(x)]$ where $\mu^0(x) = x$, and $E[\mathbf{g}^i | \mathbf{g}^{i-1} = \mu^{i-1}]$ is the expected value of $\mathbf{g}^i$ under the RBM conditional distribution $Q(\mathbf{g}^i | \mathbf{g}^{i-1})$, when the values of $\mathbf{g}^{i-1}$ are replaced by the mean-field values $\mu^{i-1}(x)$. In the case where $\mathbf{g}^i$ has binomial units, $E[\mathbf{g}^i_j | \mathbf{g}^{i-1} = \mu^{i-1}] = \text{sigm}(-b^i_j - \sum_k W^i_{jk} \mu^{i-1}_k(x))$.
- Define the network output function $f(x) = V(\mu^L(x)', 1)'$
- Iteratively minimize the expected value of $C(f(x), y)$ for pairs (x, y) sampled from $\widehat{p}$ by tuning parameters W, b, V. This can be done by stochastic gradient descent with learning rate ϵ_C, using an appropriate stopping criterion such as early stopping on a validation set.

Algorithm 6 `TrainPartiallySupervisedLayer`$(\widehat{p}, C, \epsilon_C, \epsilon_{CD}, W, b, V)$
This procedure should be called as an alternative to the loop that calls `RBMupdate` *in* `TrainUnsupervisedDBN`*, in order to train with partial supervision: perform unsupervised parameters updates with contrastive divergence, followed by greedy supervised gradient stochastic updates with respect to C, using temporary output weights V to map the hidden layer outputs to predictions.*
$\widehat{p}$ is the supervised training distribution, with samples (x, y), x being the input of the layer, and y the target for the network
C is a training criterion, a function that takes a prediction $f(x)$ and a target y and returns a scalar differentiable in $f(x)$
ϵ_{CD} is a learning rate for the stochastic gradient descent with Contrastive Divergence
ϵ_C is a learning rate for the stochastic gradient descent on supervised cost C
W is the weight matrix for the layer to train
b is the bias vector for that layer
V is a weight matrix that transforms hidden activations into predictions $f(x)$

- Define the mean-field output of the hidden layer, $\mu(x) = E[\mathbf{h}|x]$, for example $\mu(x) = \text{sigm}(-b_j - \sum_k W_{jk} x_k)$ for binomial hidden units.
- Define the layer predictive output function $f(x) = V(\mu(x)', 1)'$
- Initialize all parameters $\theta = (W, b, V)$ to 0

while not stopping criterion **do**

- sample (x, y) from $\widehat{p}$
- compute units activation (e.g. $-b - Wx$)
- using these activations, compute hidden units mean-field output $\mu(x)$
- using these activations, sample $\mathbf{h}_0$ from $Q(\mathbf{h}|x)$
- compute predictive output $f(x)$ from $\mu(x)$
- compute predictive cost C from $f(x)$ and y
- compute $\frac{\partial C}{\partial \theta}$ by standard back-propagation
- sample $\mathbf{v}_1$ from $P(\mathbf{v}|\mathbf{h}_0)$
- compute $Q(\mathbf{h}_1|\mathbf{v}_1)$
- perform supervised stochastic gradient update $\theta \leftarrow \theta - \epsilon_C \frac{\partial C}{\partial \theta}$
- $W \leftarrow W - \epsilon_{CD}(\mathbf{h}_0 x' - Q(\mathbf{h}_{1.} = 1|\mathbf{v}_1)\mathbf{v}_1')$
- $b \leftarrow b - \epsilon_{CD}(\mathbf{h}_0 - Q(\mathbf{h}_{1.} = 1|\mathbf{v}_1))$
- $c \leftarrow c - \epsilon_{CD}(\mathbf{v}_0 - \mathbf{v}_1)$

end while

Algorithm 7 `TrainGreedySupervisedDeepNet`$(\widehat{p}, C, \epsilon, L, n, W, b, V)$

Greedily train a deep network layer-wise, using a supervised criterion to optimize each layer, as if it were the hidden layer of a one-hidden-layer neural network.

$\widehat{p}$ is the supervised training distribution, with samples (x, y), x being the input of the layer, and y the target for the network with (input,target) samples (x, y)

C is a training criterion, a function that takes a network output $f(x)$ and a target y and returns a scalar differentiable in $f(x)$

ϵ is a learning rate for the stochastic gradient descent on supervised cost C

W is the weight matrix for the layer to train

b is the bias vector for that layer

V is a weight matrix that transforms top-layer hidden activations into predictions $f(x)$

- initialize $b^0 = 0$.
- define $\mu^0(x) = x$.

for $\ell = 1$ to L **do**

- initialize $b^\ell = 0$.
- initialize temporary parameter vector $c^\ell = 0$ and temporary matrix $V^\ell = 0$.
- initialize W^ℓ by sampling from uniform$(-a, a)$, with $a = 1/n^{\ell-1}$.
- define the ℓ-th hidden layer output $\mu^\ell(x) = \text{sigm}(b^\ell + W^\ell \mu^{\ell-1}(x))$.
- define the ℓ-th temporary output layer prediction $f^\ell(x) = c^\ell + V^\ell \mu^\ell(x)$

while not stopping criterion **do**

for $i = 1$ to $\ell - 1$ **do**

- compute $\mu^i(x)$ from $\mu^{i-1}(x)$.

end for

- compute $\mu^\ell(x)$ from $\mu^{\ell-1}(x)$.
- compute temporary output $f^\ell(x)$ from $\mu^\ell(x)$.
- compute the prediction error C from $f^\ell(x)$ and y.
- compute $\frac{\partial C}{\partial \omega}$, for $\omega = (W^\ell, b^\ell, c^\ell, V^\ell)$
- update layer parameters: $\omega \leftarrow \omega - \epsilon \frac{\partial C}{\partial \omega}$

end while

end for
