Visualizing Higher-Layer Features of a Deep Network

Article · January 2009

CITATIONS
698

READS
37,387

4 authors, including:

Dumitru Erhan
Google Inc.
53 PUBLICATIONS **53,324** CITATIONS
SEE PROFILE

Y. Bengio
Université de Montréal
800 PUBLICATIONS **223,374** CITATIONS
SEE PROFILE

Aaron Courville
Université de Montréal
188 PUBLICATIONS **44,979** CITATIONS
SEE PROFILE

Visualizing Higher-Layer Features of a Deep Network

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent
Dept. IRO, Université de Montréal
P.O. Box 6128, Downtown Branch, Montreal, H3C 3J7, QC, Canada
first.last@umontreal.ca

Technical Report 1341
Département d'Informatique et Recherche Opérationnelle

June 9th, 2009

Abstract

Deep architectures have demonstrated state-of-the-art results in a variety of settings, especially with vision datasets. Beyond the model definitions and the quantitative analyses, there is a need for *qualitative* comparisons of the solutions learned by various deep architectures. The goal of this paper is to find good qualitative interpretations of high level features represented by such models. To this end, we contrast and compare several techniques applied on Stacked Denoising Auto-encoders and Deep Belief Networks, trained on several vision datasets. We show that, perhaps counter-intuitively, such interpretation is possible at the unit level, that it is simple to accomplish and that the results are consistent across various techniques. We hope that such techniques will allow researchers in deep architectures to understand more of how and why deep architectures work.

1 Introduction

Until 2006, it was not known how to efficiently learn deep hierarchies of features with a densely-connected neural network of many layers. The breakthrough, by Hinton et al. (2006a), came with the realization that unsupervised models such as Restricted Boltzmann Machines (RBMs) can be used to initialize the network in a region of the parameter space that makes it easier to subsequently find a good minimum of the supervised objective. The greedy, layer-wise unsupervised initialization of a network can also be carried out by using auto-associators and related models, as shown by Bengio et al. (2007) and Ranzato et al. (2007). Recently, there has been a surge in research on training deep architectures: Bengio (2009) gives a comprehensive review.

While quantitative analyses and comparisons of such models exist, and visualizations of the first layer representations are common in the literature, one area where more work needs to be done is the qualitative analysis of representations learned beyond the first level.

Some of the deep architectures (such as Deep Belief Nets (Hinton et al., 2006a)) are associated with a generative procedure, and one could potentially use such a procedure

to gain insight into what an individual hidden unit represents. We explore one such sampling technique here. However, it is sometimes difficult to obtain samples that cover well the modes of a Boltzmann or RBM distribution, and these sampling-based visualizations cannot be applied to to other deep architectures such as those based on auto-encoders (Bengio et al., 2007; Ranzato et al., 2007; Larochelle et al., 2007; Ranzato et al., 2008; Vincent et al., 2008) or on semi-supervised learning of similarity-preserving embeddings at each level (Weston et al., 2008).

A typical *qualitative* way of comparing features extracted by a *first layer* of a deep architecture is by looking at the "filters" learned by the model, that is the linear weights in the input-to-first layer weight matrix, represented in input space. This is particularly convenient when the inputs are images or waveforms, which can be visualized. Often, these filters take the shape of stroke detectors, when trained on digit data, or edge detectors (Gabor filters) when trained on natural image patches (Hinton et al., 2006a; Hinton et al., 2006b; Osindero & Hinton, 2008; Larochelle et al., 2009). The techniques we study here also suppose that the input patterns can be displayed and are meaningful for humans, and we evaluate all of them on image data.

Our aim was to explore ways of visualizing what a unit computes in an *arbitrary layer* of a deep network. The goal was to have this visualization in the *input space* (of images), to have an efficient way of computing it, and to make it as general as possible (in the sense of it being applicable to a large class of neural-network-like models). To this end, we explore several visualization methods that allow us to gain insight into what a particular unit of a neural network represents. We compare and contrast them qualitatively on two image datasets, and we also explore connections between all of them.

The main experimental finding of this investigation is very surprising: the response of an internal unit to input images, as a function in image space, appears to be unimodal, or at least that the maximum is found reliably and consistently for all the random initializations tested. This is interesting because finding this dominant mode is relatively easy, and displaying it then provides a good characterization of what the unit does.

2 The models

We shall consider two deep architectures as representatives of two families of models encountered in the deep learning literature. The first model is a Deep Belief Net (DBN) (Hinton et al., 2006a), obtained by training and stacking three layers as Restricted Boltzmann Machines (RBM) in a greedy manner. This means that we trained a RBM with Contrastive Divergence (Hinton, 2002) on the training data, we fixed the parameters of this RBM, and then trained another RBM to model the hidden layer representations of the first level RBM. This process can be repeated to yield a deep architecture that is an unsupervised model of the training distribution. Note that it is also a generative model of the data and one can easily obtain samples from a trained model. DBNs have been described numerous times in the literature and we use them as described by (Bengio et al., 2007) and (Hinton et al., 2006a); we omit more details in favor of describing the other deep architecture.

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 denois-

ing 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. 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; Ranzato et al., 2008). The activation maximization technique presented below is applicable to any trained deep neural network, and we evaluate it on networks obtained by stacking RBMs and denoising auto-encoders.

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 $h(\mathbf{x}) = \mathrm{sigmoid}(\mathbf{b} + W\mathbf{x})$ is an ordinary neural network layer, with hidden unit biases $\mathbf{b}$, weight matrix W, and $\mathrm{sigmoid}(\mathbf{a}) = 1/(1 + \exp(-\mathbf{a}))$ (applied element-wise on a vector $\mathbf{a}$). 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 $C_i(\mathbf{x}) = \mathrm{Bernoulli}(0.5)$. The "reconstruction" is obtained from the noisy input with $\hat{\mathbf{x}} = \mathrm{sigmoid}(\mathbf{c} + W^T h(C(\mathbf{x})))$, 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 $\partial\mathrm{KL}(\mathbf{x}||\hat{\mathbf{x}}) = \sum_i \partial\mathrm{KL}(x_i||\hat{x_i})$ the sum of component-wise KL divergences between the Bernoulli probability distributions associated with each element of $\mathbf{x}$ and its reconstruction probabilities $\hat{\mathbf{x}}$: $\mathrm{KL}(\mathbf{x}||\hat{\mathbf{x}}) = -\sum_i (x_i \log \hat{x}_i + (1 - x_i) \log (1 - \hat{x}_i))$. 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.

For each unlabeled example $\mathbf{x}$, a stochastic gradient estimator is then obtained by computing $\partial\mathrm{KL}(\mathbf{x}||\hat{\mathbf{x}})/\partial\theta$ for $\theta = (\mathbf{b}, \mathbf{c}, W)$. The gradient is stochastic because of sampling example $\mathbf{x}$ and because of the stochastic corruption $C(\mathbf{x})$. Stochastic gradient descent $\theta \leftarrow \theta - \epsilon \cdot \partial\mathrm{KL}(\mathbf{x}||\hat{\mathbf{x}})/\partial\theta$ is then performed with learning rate ϵ, for a fixed number of pre-training iterations.

3 Maximizing the activation

The first idea is simple: we look for input patterns of bounded norm which maximize the *activation* of a given hidden unit[1]; since the activation function of a unit in the first layer is a linear function of the input, in the case of the first layer, this input pattern is proportional to the filter itself.

The reasoning behind this idea is that a pattern to which the unit is responding maximally could be a good first-order representation of what a unit is doing. One simple way of doing this is to find, for a given unit, the input sample(s) (from either the training or the test set) that give rise to the highest activation of the unit. Unfortunately, this still leaves us with the problem of choosing how many samples to keep for each unit and the problem of how to "combine" these samples. Ideally, we would like to find out what these samples have in common. Furthermore, it may be that only some subsets of the input vector contribute to the high activation, and it is not easy to determine which by inspection.

Note that we restricted ourselves needlessly to searching for an input pattern from *the training or test sets*. We can take a more general view and see our idea—maximizing the activation of a unit—as an *optimization* problem. Let θ denote our neural network parameters (weights and biases) and let $h_{ij}(\theta, \mathbf{x})$ be the activation of a given unit i from a given layer j in the network; h_{ij} is a function of *both* θ *and* the input sample $\mathbf{x}$. Assuming a fixed θ (for instance, the parameters after training the network), we can view our idea as looking for

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \; s.t. \; ||\mathbf{x}||=\rho} h_{ij}(\theta, \mathbf{x}).$$

This is, in general, a non-convex optimization problem. But it is a problem for which we can at least try to find a local minimum. This can be done most easily by performing simple *gradient ascent* in the input space, i.e. computing the gradient of $h_{ij}(\theta, \mathbf{x})$ and moving $\mathbf{x}$ in the direction of this gradient[2].

Two scenarios are possible: the same (qualitative) minimum is found when starting from different random initializations or two or more local minima are found. In both cases, the unit can then be characterized by the minimum or set of minima found. In the latter case, one can either average the results, or choose the one which maximizes the activation, or display all the local minima obtained to characterize that unit.

This optimization technique (we will call it "activation maximization") is applicable to any network in which we can compute the above gradients. Like any gradient descent technique, it does involve a choice of hyperparameters: the learning rate and a stopping criterion (the maximum number of gradient ascent updates, in our experiments).

4 Sampling from a unit of a Deep Belief Network

Consider a Deep Belief Network with j layers, as described in Section 2. In particular, layers $j-1$ and j form an RBM from which we can sample using block Gibbs sampling,

[1] The total sum of the input to the unit from the previous layer plus its bias.

[2] Since we are trying to *maximize* h_{ij}.

which successively samples from $p(\mathbf{h}_{j-1}|\mathbf{h}_j)$ and $p(\mathbf{h}_j|\mathbf{h}_{j-1})$, denoting by $\mathbf{h}_j$ the binary vector of units from layer j. Along this Markov chain, we propose to "clamp" unit h_{ij}, and only this unit, to 1. We can then sample inputs $\mathbf{x}$ by performing ancestral top-down sampling in the directed belief network going from layer $j-1$ to the input, in the DBN. This will produce a distribution that we shall denote by $p_j(\mathbf{x}|h_{ij}=1)$ where h_{ij} is the unit that is clamped, and p_j denotes the depth-j DBN containing only the first j layers. This procedure is similar to and inspired from experiments by Hinton et al. (2006a), where the top layer RBM is trained on the representations learned by the previous RBM *and the label* as a one-hot vector; in that case, one can "clamp" the label vector to a particular configuration and sample from a particular class distribution $p(\mathbf{x}|class=k)$.

In essence, we use the distribution $p_j(\mathbf{x}|h_{ij}=1)$ to characterize h_{ij}. In analogy to Section 3, we can characterize the unit by many samples from this distribution or summarize the information by computing the expectation $E[\mathbf{x}|h_{ij}=1]$. This method has, essentially, no hyperparameters except the number of samples that we use to estimate the expectation. It is relatively efficient provided the Markov chain at layer j mixes well (which is not always the case, unfortunately).

There is an interesting link between the method of maximizing the activation and $E[\mathbf{x}|h_{ij}=1]$. By definition, $E[\mathbf{x}|h_{ij}=1]=\int \mathbf{x}p_j(\mathbf{x}|h_{ij}=1)d\mathbf{x}$. If we consider the extreme case where the distribution concentrates at $\mathbf{x}^+$, $p_j(\mathbf{x}|h_{ij}=1)\approx\delta_{\mathbf{x}^+}(\mathbf{x})$, then the expectation is $E[\mathbf{x}|h_{ij}=1]=\mathbf{x}^+$.

On the other hand, when applying the activation maximization technique to a DBN, we are approximately [3] looking for $\arg\max_{\mathbf{x}} p(h_{ij}=1|\mathbf{x})$, since this probability is monotonic in the activation of unit h_{ij}. Using Bayes' rule and the concentration assumption about $p(\mathbf{x}|h_{ij}=1)$, we find that

$$p(h_{ij}=1|\mathbf{x})=\frac{p(\mathbf{x}|h_{ij}=1)p(h_{ij}=1)}{p(\mathbf{x})}=\frac{\delta_{\mathbf{x}^+}(\mathbf{x})p(h_{ij}=1)}{p(\mathbf{x})}$$

This is zero everywhere except at $\mathbf{x}^+$ so under our assumption, $\arg\max_{\mathbf{x}} p(h_{ij}=1|\mathbf{x})=\mathbf{x}^+$.

More generally, one can show that if $p(\mathbf{x}|h_{ij}=1)$ concentrates sufficiently around $\mathbf{x}^+$ compared to $p(\mathbf{x})$, then the two methods (expected value over samples vs activation maximization) should produce very similar results. Generally speaking, it is easy to imagine how such an assumption could be untrue because of the nonlinearities involved. In fact, what we observe is that although the samples or their average may look like training examples, the images obtained by activation maximization look more like image parts, which may be a more accurate representation of what the particular units does (by opposition to all the other units involved in the sampled patterns).

5 Linear combination of previous layers' filters

Lee et al. (2008) showed one way of visualizing what the units in the second hidden layer of a network are responding to. They made the assumption that a unit can be

[3] because of the approximate optimization and because the true posteriors are intractable for higher layers, and only approximated by the corresponding neural network unit outputs.

characterized by the filters of the previous layer to which it is most strongly connected[4]. By taking a weighted linear combination of the previous layer filters—where the weight of the filters is its weight to the unit considered—they show that a Deep Belief Network with sparsity constraints on the activations, trained on natural images, will tend to learn "corner detectors" at the second layer. Lee et al. (2009) used an extended version of this method for visualizing units of the third layer: by simply weighing the "filters" found at the second layer by their connections to the third layer, and choosing again the largest weights.

Such a technique is simple and efficient. One disadvantage is that it is not clear how to automatically choose the appropriate number of filters to keep at each layer. Moreover, by selecting only the very few most strongly connected filters from the first layer, one can potentially get a misleading picture, since one is essentially ignoring the rest of the previous layer units. Finally, this method also bypasses the nonlinearities between layers, which may be an important part of the model. One motivation for this paper is to validate whether the patterns obtained by Lee et al. (2008) are similar to those obtained by the other methods explored here.

One should note that there is indeed a link between the gradient updates for maximizing the activation of a unit and finding the linear combination of weights as described by Lee et al. (2009). Take, for instance h_{i2}, i.e. the activation of unit i from layer 2 with $h_{i2} = \mathbf{v}'\mathrm{sigmoid}(W\mathbf{x})$, with $\mathbf{v}$ being the unit's weights and W being the first layer weight matrix. Then $\partial h_{i2}/\partial \mathbf{x} = \mathbf{v}'\mathrm{diag}(\mathrm{sigmoid}(W\mathbf{x}) * (\mathbf{1} - \mathrm{sigmoid}(W\mathbf{x})))W$, where $*$ is the element-wise multiplication, diag is the operator that creates a diagonal matrix from a vector, and $\mathbf{1}$ is a vector filled with ones. If the units of the first layer do not saturate, then $\partial h_{i2}/\partial \mathbf{x}$ points roughly in the direction of $\mathbf{v}'W$, which can be approximated by taking the terms with the largest absolute value of $\mathbf{v}_i$.

6 Experiments

6.1 Data and setup

We used two datasets: the first is an extended version of the MNIST digit classification dataset, by Loosli et al. (2007), in which elastic deformations of digits are generated stochastically. We used 2.5 million examples as training data, where each example is a 28×28 gray-scale image. The second is a collection of 100000 12×12 patches of natural images, generated from the collection of whitened natural image patches by Olshausen and Field (1996).

The visualization procedures were tested on the models described in Section 2: Deep Belief Nets (DBNs) and Stacked Denoising Auto-encoders (SDAE). The hyper-parameters are: unsupervised and supervised learning rates, number of hidden units per layer, and the amount of noise in the case of SDAE; they were chosen to minimize

[4] i.e. whose weight to the upper unit is large in magnitude

the classification error on MNIST[5] or the reconstruction error[6] on natural images, for a given validation set. For MNIST, we show the results obtained after unsupervised training only; this allows us to compare all the methods (since we cannot sample from a DBN after supervised fine-tuning). For the SDAE, we used salt and pepper noise as a corruption technique, as opposed to the zero-masking noise described by Vincent et al. (2008): such noise seems to better model natural images. For both SDAE and DBN we used a Gaussian input layer when modeling natural images; these are more appropriate than the standard Bernoulli units, given the distribution of pixel grey levels in such patches (Bengio et al., 2007; Larochelle et al., 2009).

In the case of activation maximization (Section 3), the procedure is as follows for a given unit from either the second or the third layer: we initialize $\mathbf{x}$ to a vector of 28×28 or 12×12 dimensions in which each pixel is sampled independently from a uniform over $[0; 1]$. We then compute the gradient of the activation of the unit w.r.t. $\mathbf{x}$ and make a step in the gradient direction. The gradient updates are continued until convergence, i.e. until the activation function does not increase by much anymore. Note that after each gradient update, the current estimate of $\mathbf{x}^*$ is re-normalized to the average norm of examples from the respective dataset[7]. Interestingly, the same optimal value (i.e. the one that seems to maximize activation) for the learning rate of the gradient ascent works for all the units from the same layer.

Sampling from a DBN is done as described in Section 4, by running the randomly-initialized Markov chain and top-down sampling every 100 iterations. In the case of the method described in Section 5, the (subjective) optimal number of previous layer filters was taken to be 100.

6.2 Activation Maximization

We begin by the analysis of the **activation maximization** method. Figures 1 and 2 contain the results of the optimization of units from the 2nd and 3rd layers of a DBN and an SDAE, along with the first layer filters. Figure 1 shows such an analysis for MNIST and Figure 2 shows it for the natural image data.

To test the dependence of this gradient ascent on the initial conditions, 9 different random initializations were tried. The retained "filter" corresponding to each unit is the one (out of the 9 random initializations) which maximizes the activation. In the same figures we also show the variations found by the different random initializations for a given unit from the 3rd layer. **Surprisingly, most random initializations yield roughly the same prominent input pattern.** Moreover, we measured the maximum

[5]We are indeed choosing our hyperparameters based on the supervised objective. This objective is computed by using the unsupervised networks as initial parameters for supervised backpropagation. We chose to select the hyperparameters based on the classification error because for this problem we do have an objective criterion for comparing networks, which is not the case for the natural image data.

[6]For RBMs, the reconstruction error is obtained by treating the RBM as an auto-encoder and computing a deterministic value using either the KL divergence or the MSE, as appropriate. The reconstruction error of the first layer RBM is used for model selection.

[7]There is no constraint that the resulting values in $\mathbf{x}^*$ be in the domain of the training/test set values. For instance, we experimented with making sure that the values of $\mathbf{x}^*$ are in $[0; 1]$ (for MNIST), but this produced worse results. On the other hand, the goal is to find a "filter"-like result and a constraint that this "filter" is strictly in the same domain as the input image may not be necessary.

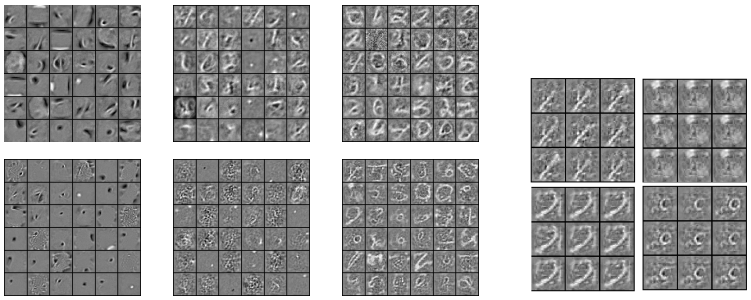
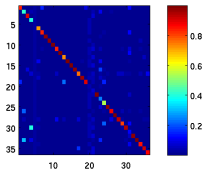


Figure 1: Activation maximization applied on MNIST. On the left side: visualization of 36 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a DBN (top) and SDAE (bottom), using the technique of maximizing the activation of the hidden unit. On the right side: 4 examples of the solutions to the optimization problem for units in the 3rd layer of the SDAE, from 9 random initializations.

values for the activation function to be quite close to each other (not shown). Such results are relatively surprising, given that, generally speaking, the activation function of a third layer unit is a highly non-convex function of its input. Therefore, either we are consistently lucky or, at least in this particular case (a network trained on MNIST digits or natural images), the activation functions of the units tend to be more "unimodal".

To further test the robustness of the activation maximization method, we perform a sensitivity analysis in order to test whether the units are selective to these patterns found by the optimization routine, and whether these patterns strongly activate other units as well. The figure on the right shows the post-sigmoidal activation of unit j (columns) when the input to the network is the "optimal" pattern i (rows), found by our gradient procedure for unit i, normalized across columns in order to eliminate the effect of units that are activated for very many patterns in general. The strong values on the diagonal suggest that the results of the optimization have uncovered patterns that are mostly specific to a particular unit.



One important point is that, qualitatively speaking, the filters at the 3rd layer look interpretable and quite complex. For MNIST, some look like pseudo-digits. In the case of natural images, we can observe grating filters at the second layer of DBNs and complicated units that detect, for instance, corners at the second and third layer of SDAE; some of the units have the same characteristics that we would associate with so-called complex cells. It suggests that higher level units *did indeed learn* meaningful combinations of lower level features.

Note that the first layer filters obtained by the SDAE when trained on natural images are Gabor-like features. It is interesting that in the case of DBN, the filters that minimized the reconstruction error[8], i.e. those that are pictured in Figure 2 (top-left corner), do not have the same low-frequency and sparsity properties like the ones found

[8]Which is only a proxy for the actual objective function that is minimized by a stack of RBMs.

by the first-level denoising auto-encoder[9]. Yet at the second layer, the filters found by activation maximization are a mixture of Gabor-like features and grating filters. This shows that appearances can be deceiving: we might have dismissed the RBM whose weights are shown in Figure 2 as a bad model of natural images had we looked only at the first layer filters, but the global qualitative assessment of this model, which includes the visualization of the second and third layers, points to the fact that the 3-layer DBN is in effect learning something quite interesting.

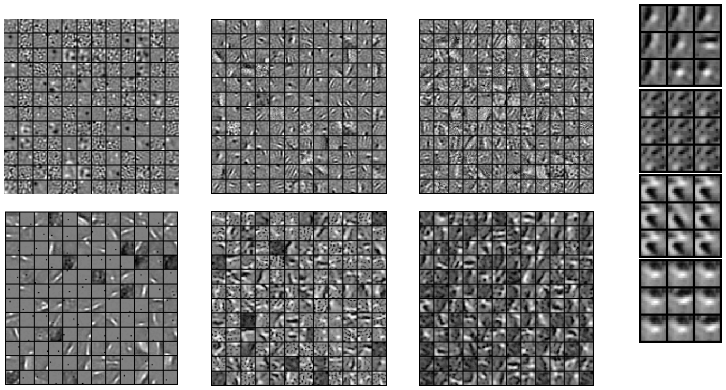


Figure 2: On the left side: Visualization of 144 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a DBN (top) and an SDAE (bottom), using the technique of maximizing the activation of the hidden unit. On the right side: 4 examples of the solutions to the optimization problem for units in the 3rd layer of the SDAE, subject to 9 random initializations.

6.3 Sampling a unit

Of note is the fact that unlike the results of the activation maximization method, the samples are much more likely to be part of the underlying distribution of examples (digits or patches). The activation maximization method seems to produce *features* and it is up to us to decide which examples would "fit" these features; the sampling method produces *examples* and it lets us decide which features these examples have in common. In this respect, the two techniques serve complementary purposes.

6.4 Comparison of methods

In Figure 4, we can see a comparison of the three techniques, including the **linear combination method**. The methods are tested on the second layer of a DBN trained on MNIST. In the above, we noted links between the three techniques. The experiments show that many of the filters found by the three methods share some features, but have a different nature. Unfortunately, we do not have an objective measure that would allow us to compare the three methods, but visually the activation maximization

[9]It *is* possible to obtain Gabor-like features with RBMs—work by Osindero and Hinton (2008) shows that—but in our case these filters were never those that minimized the reconstruction error of an RBM. This points to a larger issue: it appears that using different learning rates for Contrastive Divergence learning will induce features that are *qualitatively different*, depending on the value of the learning rate.

We now turn to the **sampling technique** described in Section 4. Figure 3 shows samples obtained by clamping a second layer unit to 1; both MNIST and natural image patches are considered. In the case of natural image patches, the distributions are roughly unimodal, in that the samples are of the same pattern, for a given unit. For MNIST, the situation is slightly more delicate: there seem to be one or two modes for each unit. The *average* input (the expectation of the distribution), as seen in Figure 4, then looks like a digit or a superposition of two digits.

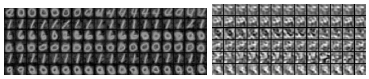


Figure 3: Visualization of 6 units from the second hidden layer of a DBN trained on MNIST (left) and natural image patches (right). The visualizations are produced by sampling from the DBN and clamping the respective unit to 1. Each unit's distribution is a row of samples; the mean of each row is in the first column of Figure 4 (left).

method seems to produce more interesting results: by comparison, the average samples from the DBN are almost always in the shape of a digit (for MNIST), while the linear combination method seems to find only parts of the features that are found by activation maximization, which tends to find sharper patterns.

6.5 Limitations

We tested the activation maximization procedure on image patches of 20×20 pixels (instead of 12×12) and found that the optimization does not converge to a single global minimum. Moreover, the input distribution that is sampled with the units clamped to 1 has many different modes and its expectation is not meaningful or interpretable anymore. We posit that these methods break down because of the complexity of the input distribution: both MNIST and 12×12 image patches are relatively simple distributions to model and this could be the reason these methods work in the first place. It is perhaps unrealistic to expect that as we scale the datasets to larger and larger images, one could still find a simple representation of a higher layer unit. We should note, however, that there is a recent trend of developing convolutional versions of deep architectures (Kavukcuoglu et al., 2009; Lee et al., 2009; Desjardins & Bengio, 2008): it is likely that one will be able to apply the same techniques in that scenario and still be able to recover good visualizations, even with large inputs.

7 Conclusions and Future Work

We started from a simple premise: to better understand the solution that is learned and represented by a deep architecture, by investigating the response of individual units in the network. Like the analysis of individual neurons in the brain by neuroscientists, this approach has limitations, but we hope that such visualization techniques can help understand the nature of the functions learned by the network.

We presented three simple techniques: activation maximization and sampling from a unit are both new (to the best of our knowledge), while the linear combination technique had been previously published. We showed the intuitive similarities between

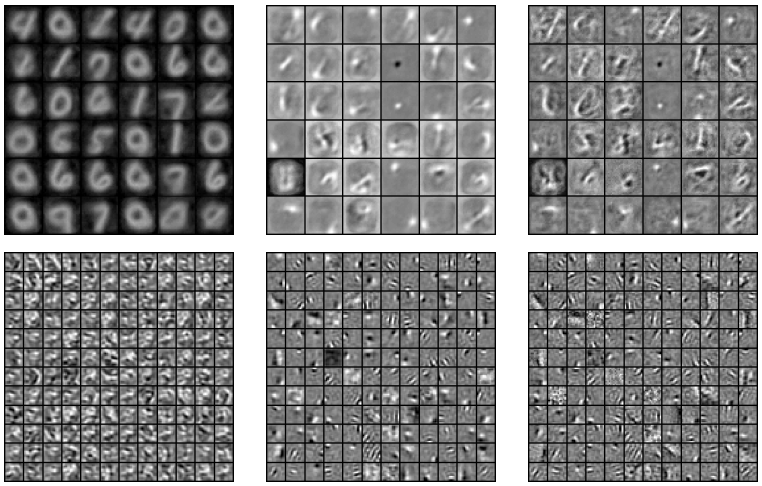


Figure 4: Visualization of 36 units from the second hidden layer of a DBN trained on MNIST (top) and 144 units from the second hidden layer of a DBN trained on natural image patches (bottom). Left: sampling with clamping, Centre: linear combination of previous layer filters, Right: maximizing the activation of the unit. Black is negative, white is positive and gray is zero.

them and compared and contrasted them on two well-known datasets. Our results confirm the intuitions that we had about the hierarchical representations learned by deep architectures: namely that the higher layer units represent features that are (meaningfully) more complicated and that correspond to combinations of features of the lower layers. We have also found that the two deep architectures considered learn quite different features.

The same procedures can be applied to the weights obtained after *supervised* learning and the observations are similar: convergence occurs and features seem more complicated at higher layers. In the future, we would like to use such visualization tools to compare the features learned by these networks after supervised learning, in order to better understand the differences in test error and to further understand the influence of unsupervised initialization in training deep models. We will also extend our results by performing experiments with other datasets and models, such as Convolutional Networks applied to higher-resolution natural images. Finally, we would like to compare the behaviour of higher level units in a deep network to features that are presumed to be encoded by the higher levels of the visual cortex.

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2, Issue 1*.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems 19 (NIPS'06)* (pp. 153–160). MIT Press.

Desjardins, G., & Bengio, Y. (2008). *Empirical evaluation of convolutional RBMs for vision* (Technical Report 1327). Dept. IRO, U. Montreal.

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

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

Hinton, G. E., Osindero, S., Welling, M., & Teh, Y. (2006b). Unsupervised discovery of non-linear structure using contrastive backpropagation. *Cognitive Science*, *30*.

Kavukcuoglu, K., Ranzato, M., Fergus, R., & LeCun, Y. (2009). Learning invariant features through topographic filter maps. *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'09)*. IEEE.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. *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. *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)* (pp. 473–480). Corvallis, OR: ACM.

Lee, H., Ekanadham, C., & Ng, A. (2008). Sparse deep belief net model for visual area V2. In J. Platt, D. Koller, Y. Singer and S. Roweis (Eds.), *Advances in neural information processing systems 20 (nips'07)*. Cambridge, MA: MIT Press.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman (Eds.), *Proceedings of the twenty-sixth international conference on machine learning (icml'09)*. Montreal (Qc), Canada: ACM.

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.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*, 607–609.

Osindero, S., & Hinton, G. E. (2008). Modeling image patches with a directed hierarchy of markov random field. *Advances in Neural Information Processing Systems 20 (NIPS'07)* (pp. 1121–1128). Cambridge, MA: MIT Press.

Ranzato, M., Boureau, Y.-L., & LeCun, Y. (2008). Sparse feature learning for deep belief networks. *Advances in Neural Information Processing Systems 20 (NIPS'07)* (pp. 1185–1192). Cambridge, MA: MIT Press.

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems 19 (NIPS'06)* (pp. 1137–1144). MIT Press.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)* (pp. 1096–1103). ACM.

Weston, J., Ratle, F., & Collobert, R. (2008). Deep learning via semi-supervised embedding. *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)* (pp. 1168–1175). New York, NY, USA: ACM.