Understanding Representations Learned in Deep Architectures

Dumitru Erhan, Aaron Courville, and Yoshua Bengio
Dept. IRO, Université de Montréal
P.O. Box 6128, Downtown Branch, Montreal, H3C 3J7, QC, Canada
first.last@umontreal.ca
Technical Report 1355
Département d'Informatique et Recherche Opérationnelle

October 19th, 2010

Abstract

Deep architectures have demonstrated state-of-the-art performance in a variety of settings, especially with vision datasets. Deep learning algorithms are based on learning several levels of *representation* of the input. Beyond test-set performance, there is a need for *qualitative* comparisons of the solutions learned by various deep architectures, focused on those learned representations. One of the goals of our research is to improve tools for finding good qualitative interpretations of high level features learned by such models. We also seek to gain insight into the invariances learned by deep networks. To this end, we contrast and compare several techniques for finding such interpretations. We applied our techniques on Stacked Denoising Auto-Encoders and Deep Belief Networks, trained on several vision datasets. We show that consistent filter-like interpretation is possible and simple to accomplish at the unit level. The tools developed make it possible to analyze deep models in more depth and accomplish the tracing of *invariance manifolds* for each of the hidden units. 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 good minima of the supervised objective, i.e., which give good generalization error. The greedy, layer-wise unsupervised initialization of a network can also be carried out by using auto-associators and

related models (Bengio et al., 2007; Ranzato et al., 2007). Recently, there has been a surge in research on training deep architectures: Bengio (2009) gives a comprehensive review.

There exists a flurry of ideas on how pre-training should be done, how to better train deep models and how to, in general, learn better hierarchical representations of data. There has also been some progress in better understanding the effect of unsupervised pre-training and its role as a regularizer (Erhan et al., 2010). And while *quantitative* analyses and comparisons of various strategies, models and techniques 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. Qualitative analysis would bring us insights into the models used, and would allow us to compare them beyond merely measuring performance on a held-out dataset.

We want to understand what the models have learned: what features of the data models have captured and which ones they have not. Answers to that question would help tackle issues that are potentially difficult to address with a purely quantitative approach. For instance, what is the difference between the representations learned by a Deep Belief Network (DBN) and a Stacked Denoising Auto-Encoder (SDAE), when both models perform similarly on the same test set? It would also be helpful in providing evidence to support the hypothesis that deep representations are capturing and disentangling interesting features of the data.

To better understand what models learn, we set as an aim the exploration of ways to visualize what a unit activates in an *arbitrary layer* of a deep network. The goal is to have this visualization in the *input space* (of images), while remaining computationally efficient, and to make it as general as possible (in the sense of it being applicable to a large class of neural-network-like models).

For a first layer unit, given its quasi-linear response (ignoring the sigmoidal nonlinearity), a typical visualization is simply showing in the input space (e.g. as an image) the input weights of the unit, also called the filters or "receptive fields". This is particularly convenient when the inputs are images or waveforms, which can be visually interpreted by humans. 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,b; Osindero and Hinton, 2008; Larochelle et al., 2009).

For higher-level (deeper) layers, one could approach the problem from a few different angles. One approach is to devise sampling techniques. For instance, Deep Belief Nets by Hinton et al. (2006a) have an associated generative procedure, and one could potentially use such a procedure to gain insight into what an individual hidden unit represents; we introduce such an approach in this work. Note that methods that rely on sampling will likely produce output similar to examples from the training distribution and one might need to further process the samples in order into get a picture of what the unit represents. A second approach, introduced in this paper, is inspired by the idea of maximizing the response of a given unit. One of the experimental findings of this investigation is quite surprising: despite its limitations (local minima), this method was able to find coherent filter-like representations for deeper units. A third approach, by Lee et al. (2008), produces a filter-like representation for deeper units from a linear combinations of lower-level filters. Our results appear consistent across various

datasets and techniques.

In this paper, compare and contrast these techniques qualitatively on several image datasets, and we also explore connections between all of them. Even if we obtain a "filter"-like representation of a unit from a deep layer, it does not tell us the whole picture, because of the nonlinear relationship between the input and the unit response. One way of getting more insight into such a nonlinear unit is by testing its invariance against a specific set of variations in the input, e.g. rotations (Goodfellow et al., 2009). We argue in this paper that it is useful to seek a *set* of invariances, or *invariance manifolds* for each of these units. In particular, we explore a general method that is not tied to a specific list of invariances. Such an invariance analysis could be a way to gain more insight into what the units of those layers capture. A contribution of this paper is the introduction of a few general tools that make the feature and invariance analysis of deeper layers possible.

2 Previous work

We briefly go over previous attempts at solving the visualization and invariance problem, in contexts similar to ours.

2.1 Linear combination of previous units

Lee et al. (2008) showed one way of visualizing the activation pattern of units in the second hidden layer of a Deep Belief Network (Hinton et al., 2006a). They made the assumption that a unit can be characterized by the filters of the previous layer to which it is most strongly connected[1]. 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, 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 when there is not a small group of large weights but rather many smaller and similar-magnitude weights into a unit. 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.

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

2.2 Output unit sampling

Consider a Deep Belief Network with several layers. A typical scenario is where the top layer is an RBM that sees as its visible input a concatenation of the representation produced by lower levels and a one-hot vector indicating the class label. In that case, one can "clamp" the label vector to a particular configuration and sample from a particular class distribution $p(\mathbf{x}|class = k)$. Such a procedure, first described by Hinton et al. (2006a), makes it possible to "visualize" output units, as distributions in the input space. As described in section 4.1, such a procedure can be extended to an arbitrary unit in the network.

It is sometimes difficult to obtain samples that cover well the modes of a Boltzmann Machine or RBM distribution, and these sampling-based visualizations cannot be applied 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). Moreover, sampling produces a *distribution* for each unit: figuring out relevant statistics of that distribution (e.g., the modes) is potentially not straightforward.

2.3 Optimal stimulus analysis for quadratic forms

Berkes and Wiskott (2006) start with an idea, inspired by neurophysiological experiments, of computing the optimal excitatory (and inhibitory) stimulus, in the for quadratic functions of the input, which are learned using Slow Feature Analysis (SFA). The limitation to quadratic forms of the input makes it possible to find the optimal stimulus, i.e. the one maximizing the activation, relatively easily.

Berkes and Wiskott (2006) also consider an invariance analysis of the optimal stimulus, whereby one finds transformations of the input to which the quadratic form is most insensitive. This method of finding invariance is using the geodetic path, meaning the path along a sphere (norm constraint, in this case), which has the smallest "acceleration"[2] as possible.

These ideas are the closest in spirit to the work that we introduce in this paper, related to maximizing the response of a given unit. The key differences, on which we elaborate in section 5, are that we consider general nonlinear functions of the input (and not just quadratic forms) and our invariance analysis is a more direct and more non-local application of the idea that the directions of invariance should be the ones in which the function value (activation) drops least for such general nonlinear functions.

3 The models

For our analysis, 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 of Restricted Boltzmann Machines (RBM) in a greedy manner. This means that we

[2] of the considered function, in this case the activation function.

trained an RBM with Contrastive Divergence (Hinton, 2002), 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, a generative model of the data from which one can easily obtain samples from a trained model. DBNs have been described numerous times in the literature, please refer to Bengio (2009) and Hinton et al. (2006a) for further details.

The second model, introduced by Vincent et al. (2008), is the so-called Stacked Denoising Auto-Encoder (SDAE). It borrows the greedy principle from DBNs, but uses denoising auto-encoders as a building block for unsupervised modelling. 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, which 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).

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 representation learned by 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 randomly set $C_i(\mathbf{x}) = x_i$ or 0. A fixed-size random subset of $\mathbf{x}$ is 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. When training denoising auto-encoders 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 by $\mathrm{KL}(\mathbf{x}||\hat{\mathbf{x}}) = \sum_i \mathrm{KL}(x_i||\hat{x_i})$ 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 unlabelled 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 the 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.

4 How to obtain filter-like representations for deep units

We shall start our analysis by introducing the tools to obtain a filter-like representation for units belonging to a deep layer.

4.1 Sampling from a unit of a Deep Belief Network

Consider a Deep Belief Network with j layers, as described in section 3. In particular, layers $j-1$ and j form an RBM from which we can sample using block Gibbs sampling, 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; as mentioned in section 2.2, this procedure is similar to experiments done by Hinton et al. (2006a) for output units. This produces 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.

In essence, with this method, we use the distribution $p_j(\mathbf{x}|h_{ij}=1)$ to characterize h_{ij}. We can characterize the unit by 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, as illustrated previously (Tieleman and Hinton, 2009; Desjardins et al., 2010).

Note that this method is only applicable to models from which one can (efficiently) sample and this is a rather important restriction if one's goal is to come up with general methods for inspecting such deep architectures; for instance, it cannot be be applied architectures 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).

4.2 Maximizing the activation

We introduce a new idea: we look for input patterns of bounded norm which maximize the *activation*[3] of a given hidden unit; since the activation 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, i.e., $\mathbf{x} \cdot w$ is maximized for $\mathbf{x} \propto w$ (keeping $||\mathbf{x}||$ fixed).

The reasoning behind this idea is that a pattern to which the unit is responding maximally could be a good initial representation of what a unit is doing[4]. One simple way of doing this is to find, for a given unit, the input samples (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

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

[4]This is the reasoning for visualizing first-layer filters in the input space, too: they are the inputs to which the unit responds maximally.

what these samples have in common, i.e. to be able to synthesize a representation from them. Furthermore, it may be that only some elements of the input vector contribute to the high activation, and it may not be easy to determine the relevant elements simply by inspection.

Note that we restricted ourselves needlessly to searching for an input pattern from *the training or test sets*, or simply from the set of all valid patterns. We can take a more general view and *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 formalize this approach as searching 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*[5] 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.

Two scenarios are possible after the optimization converges: 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 (which we call "activation maximization", or AM) 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: in particular, the learning rate and a stopping criterion (the maximum number of gradient ascent updates, in our experiments).

4.3 Connections between methods

There is an interesting link between the method of maximizing the activation and the sampling method from section 4.1. 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 (AM) technique to a DBN, we are approximately [6] looking for $\arg\max_{\mathbf{x}} p(h_{ij} = 1|\mathbf{x})$, since this probability is monotonic in the (pre-sigmoid) activation of unit h_{ij}. Using Bayes' rule and the concentration assumption about $p(\mathbf{x}|h_{ij} = 1)$, we find that

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

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

[6] 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.

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 AM) 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 AM look more like **image parts**, which may be a more accurate representation of what the particular units do (by opposition to all the other units involved in the sampled patterns). This subtlety is key and it highlights the ways in which they are different and complementary.

There is also 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 $\mathbf{v}_i$.

4.4 First investigations into visualizing upper layer units

We shall begin with an investigation into the feasibility of using these methods for our stated purpose (obtaining informative filter-like representations). In the course of these experiments, we will also be able to compare these methods and observe their relative merits in action. More importantly, these experiments will build a basis for our explorations of invariance manifolds in the latter sections.

We used three datasets to validate our hypotheses:

- 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.
- A collection of $100,000$ greyscale 12×12 patches of natural images, generated from the collection of whitened natural image patches by Olshausen and Field (1996).
- *Caltech Silhouettes*, a simplified version of the Caltech-101 dataset (Fei-Fei et al., 2004), in which the shape of the target object was extracted and the entire image was binarized into a foreground and a background (Marlin et al., 2009). The dataset contains approximately 4,100 images of size 28x28 from 101 categories, with at least 20 and at most 100 examples from each class[7].

[7] The data can be downloaded from `http://people.cs.ubc.ca/~bmarlin/data/index.shtml`

The visualization procedures were tested on the models described in section 3: 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 the classification error on MNIST and Caltech Silhouettes, respectively[8] or the reconstruction error[9] on natural images, for a given validation set. For MNIST and Caltech Silhouettes, we show the results obtained after unsupervised training only; this allows us to compare all the methods (since it does not make sense to sample from a DBN after the supervised fine-tuning with backpropagation stage). For the SDAE applied on natural images, we used salt and pepper noise as a corruption technique, as opposed to the zero-masking noise described by Vincent et al. (2008): such symmetric noise seems to work better with natural images. For the DBN we used a Gaussian input layer when modelling 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 AM (section 4.2, Activation Maximization), 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 does not increase faster than a threshold rate. 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[10]. 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. Finally, 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 4.1, by running the randomly-initialized Markov chain and top-down sampling every 100 Gibbs steps. In the case of the method described in section 2.1, the (subjective) optimal number of previous layer filters was taken to be 100.

Activation Maximization We begin by the analysis of the **activation maximization** method (AM). Figures 1 and 2 contain the results of the optimization of units

[8] We are 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.

[9] 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.

[10] Such a procedure is essentially a stochastic gradient method with projection to the constraint at each step. It is possible to use better and more complicated optimization methods—such as conjugate gradient—but this adds unnecessary complexity (because of the constraint) and, in our experiments, did not lead to different conclusions.

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, while Figure 2 shows it for the natural image data and Caltech Silhouettes.

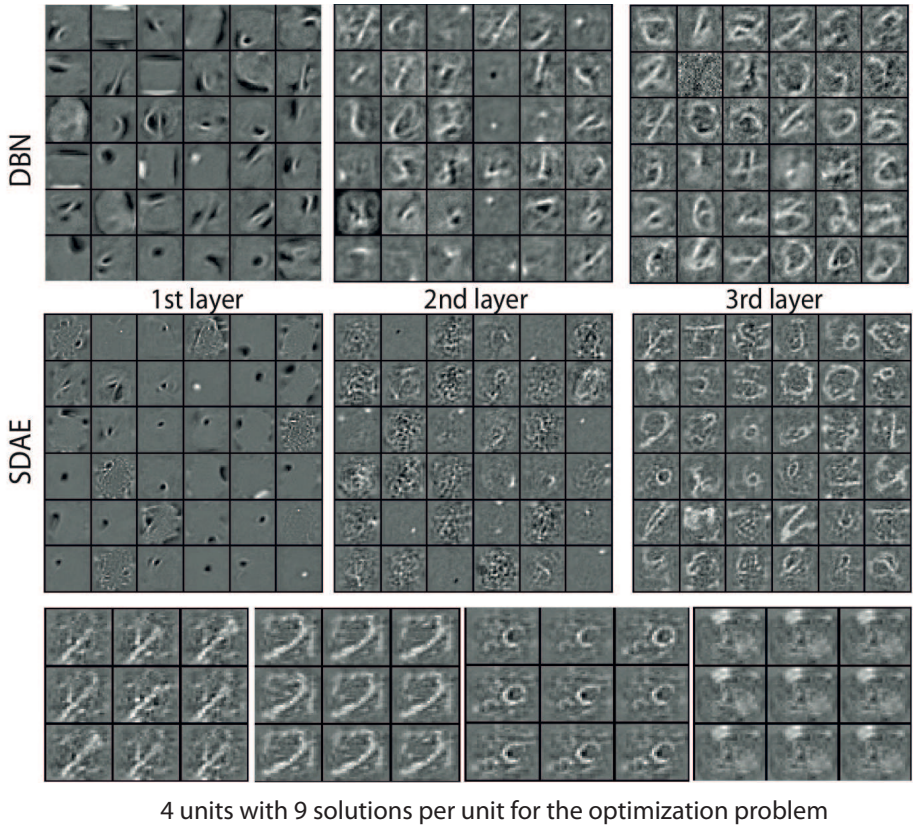

Figure 1: *Activation maximization (AM) applied on MNIST.* First two rows*: visualization of 36 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a DBN (top) and SDAE (middle), using the technique of maximizing the activation of the hidden unit.* Bottom row*: 4 examples of the solutions to the optimization problem for units in the 3rd layer of the SDAE, from 9 random initializations.*

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 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 we are not sampling from the whole space, or, at least in these particular cases (a network trained on MNIST digits, Caltech Silhouettes, or natural images), the activation functions of the units tend to be more "unimodal".

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 V2-area units (Lee et al., 2008). For Caltech Silhouettes, a few of the units look like whole-object class detectors (faces, for instance), but most seem to simply encode for the presence or absence of parts of objects (likely meaning that the third layer units have managed to learn a decomposition of the input space features that is not as simple as just whole-object-class detection). Such results also suggest that higher level units do 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 the DBN, the filters that minimized the reconstruction error[11], 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 by the first-level denoising auto-encoder[12]. 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. Such a result suggests that qualitative model comparison (between SDAE and DBNs in this case) cannot rely entirely on first-layer filter visualizations.

Sampling a unit We now turn to the **sampling technique** described in section 4.1. 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[13]. The *average* input (the expectation of the distribution), as seen in Figure 4, then looks like a digit or a superposition of two digits.

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

[12] 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.

[13] This result was obtained with multiple restarts and 20,000 Gibbs steps

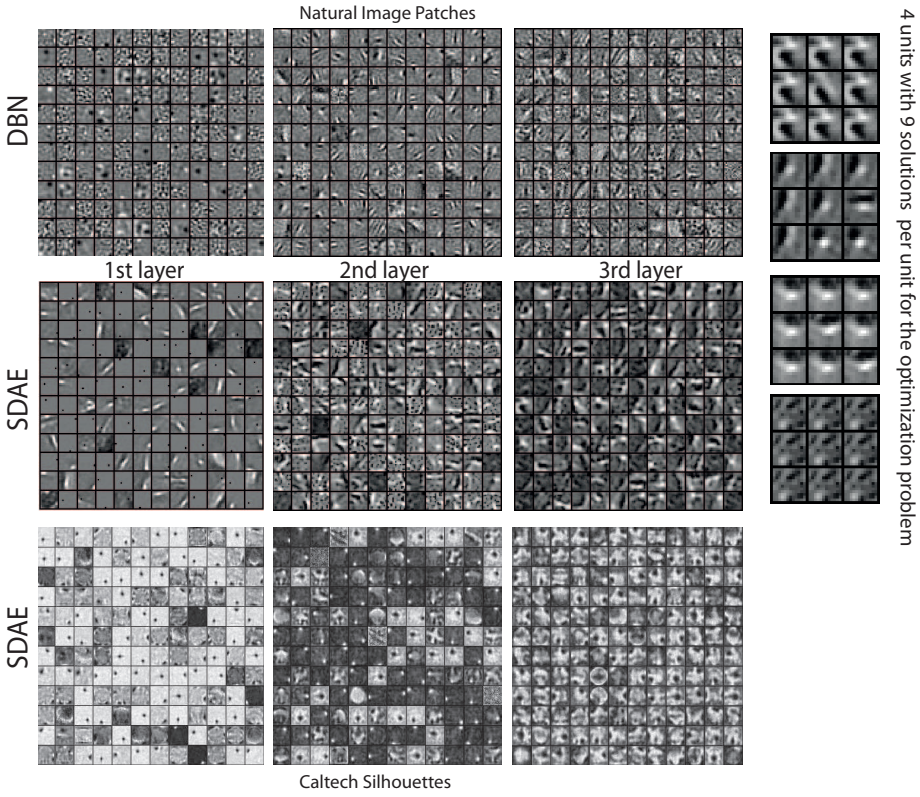

Figure 2: *Activation Maximization (AM) applied on Natural Image Patches (top and middle row) and Caltech Silhouettes (bottom row). Visualization of 144 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a DBN (top row) and an SDAE (middle and bottom rows), using the technique of maximizing the activation of the hidden unit.* In the 4th column*: 4 examples of the solutions to the optimization problem for units in the 3rd layer of the SDAE, subject to 9 random initializations, for natural images.*

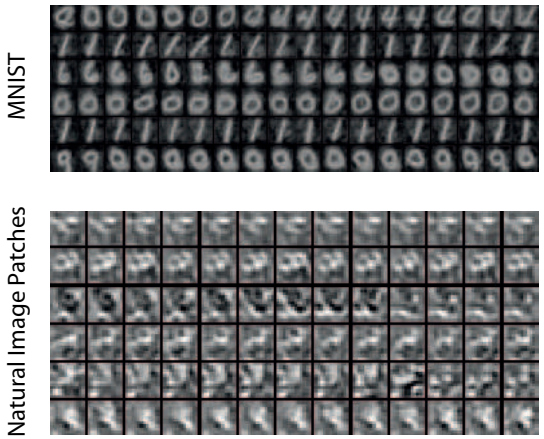

Figure 3: *Visualization of 6 units from the second hidden layer of a DBN trained on MNIST (top) and natural image patches (bottom). The visualizations are produced by sampling from the DBN and clamping the respective unit to 1. Each unit's distribution is a row of samples; the mean of each row is in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

Note that unlike the results of AM, the samples are much more likely to be part of the underlying distribution of examples (digits or patches). AM 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 leaves it to us decide which features these examples have in common. In this respect, the two techniques serve complementary purposes.

Comparison of methods In Figure 4, we can see a comparison of the three techniques: activation maximization, hidden unit sampling, and the **linear combination method**, introduced by Lee et al. (2008) and as described in section 2.1. 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 some differences as well. In general, linear combination of previous layer filters and AM were quite similar, highlighting parts, whereas sampling produced full examples.

Unfortunately, we do not have an objective measure that would allow us to compare the three methods, but visually we believe that AM produces more interesting and useful 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 AM, which tends to find sharper patterns.

AM is applicable to a very large class of models, is conceptually simple and produces high quality visualizations. Moreover, the technique lends itself to easy, but quite

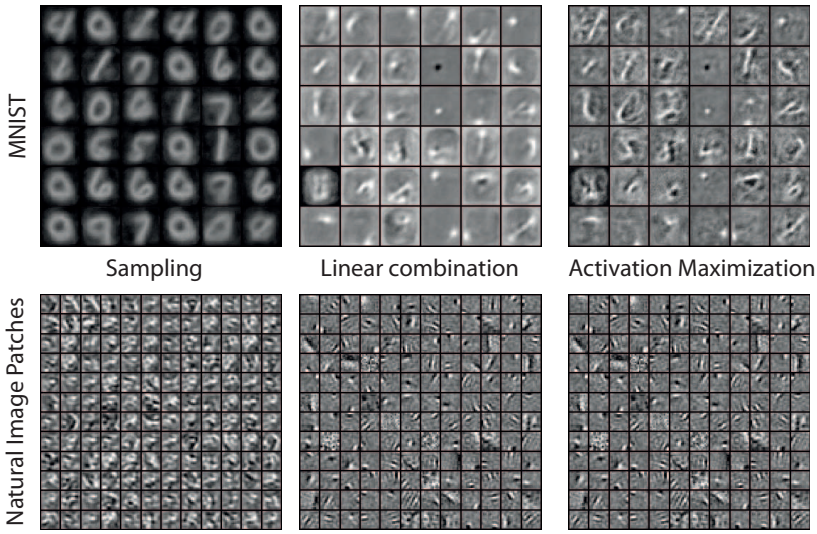

Figure 4: *Visualization of 36 units from the second hidden layer of a DBN trained on MNIST (top) and 144 units from the second hidden layer of a DBN trained on natural image patches (bottom). Left: sampling with clamping, Centre: linear combination of previous layer filters, Right: maximizing the activation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

powerful extensions, as we shall explore next.

5 Uncovering Invariance Manifolds

Thus far our goal has been to obtain a filter-like representation for each unit of the upper layers. Obtaining such filters is an interesting development and it allows us to see that upper layer units correspond to more complicated filters (sometimes even "template detectors") and verify some hypotheses that we had about deep architectures: namely that they learn to model interesting features at higher levels, that units at those levels correspond to more complicated V2-area like units etc. However, such filter-like representations only characterize a point in the input space: they don't really describe the invariances captured by each unit or each layer. The second part of our inquiry will address this issue.

A simple approach to solving this problem is by extending our activation maximization approach to computing some second order visualization. One way was presented in section 2.3, by Berkes and Wiskott (2006)[14]: compute geodetic paths (paths on the norm constraint / sphere), starting at the maximum of the activation function, which have the smallest rate of change. Another solution is to compute the Hessian at the local maximum and analyze the directions of principal invariance, corresponding to the eigenvectors of the Hessian with the smallest eigenvalues, by moving in the direction of those eigenvectors (starting from the optimum), while remaining on the norm sphere. For quadratic forms and in the context of Slow Feature Analysis, such an approach seemed to be fruitful (Berkes and Wiskott, 2002, 2006).

Our attempts at replicating the latter analysis in the context of AM and arbitrary units in the deep layers were not as successful: the eigenvectors point in directions that did not reveal useful insights, as far as we could tell. Our intuition is that such directions are really a very local measure around the maximum and may not be meaningful farther away from it. This locality effect is present in the geodesic path method of Berkes and Wiskott (2006), where the authors suggest that this method is only applicable in "a small neighbourhood" of the maximum. We would like a method that would trace an invariance manifold that corresponds to the unit, and we want this manifold to be less local (with respect to the maximum found via AM). Ideally, we would like to see what pattern of activations it is most invariant to or what manifold this unit "traces" in the input space. Finally, our intuition suggests that these directions of invariance should correspond, roughly speaking, to the changes of the optimum that produce the smallest decrease in the activation value, and we would like a more direct way of achieving this.

5.1 Invariance Manifolds

A simple way of achieving such goals is to start with the result given to us by AM and move as far as possible from it while keeping the activation as large as possible. Formally, let $\mathbf{x}_{opt}$ be the (best) local optimum found by AM for a given unit. Then we re-formulate our optimization problem as follows:

[14] for quadratic functions of the input

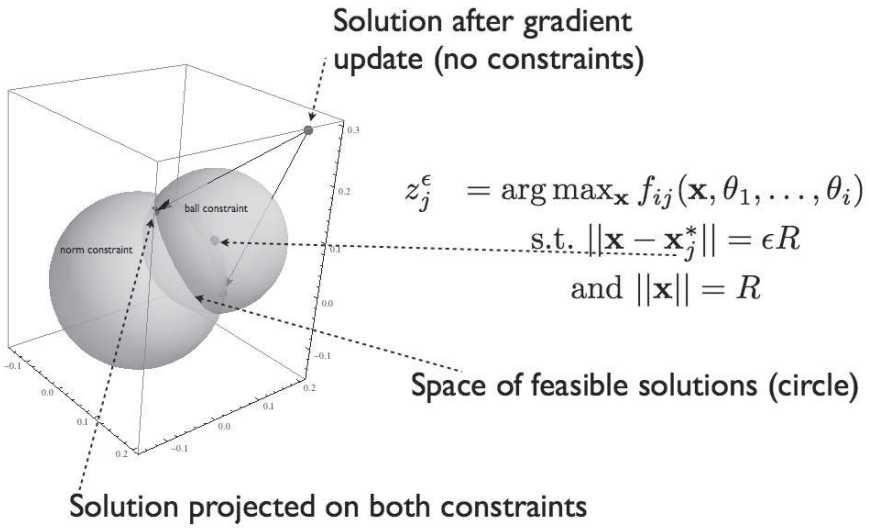

Figure 5: *Illustration of the invariance manifold tracing technique in 3D.* $\mathbf{x}_j^*$ *is the activation maximization result for unit* j*,* R *is the average norm of our inputs, and* εR *is the distance from* $\mathbf{x}_j^*$ *that we want our solutions to be. After each gradient step (towards maximizing* f_{ij}*), we project the current solution such that it satisfies the constraints; there are two such projections possible—for the next iteration of the optimization problem, we choose the one with the highest activation value.*

$$\mathbf{x}^*_\varepsilon = \arg \max_{\mathbf{x}\ s.t.\ ||\mathbf{x}||=\rho \text{ and } ||\mathbf{x}-\mathbf{x}_{opt}||=\varepsilon\rho} h_{ij}(\theta, \mathbf{x}).$$

where $0 \leq \varepsilon \leq 2$. By varying ε we can construct a one-dimensional manifold—represented by the solutions $\mathbf{x}^*_\varepsilon$ in increasing order of ε—-that has our desired properties[15].

Note that, as before, we require our solutions $\mathbf{x}^*_\varepsilon$ to be of some fixed norm as well (ρ, as before); removing such a constraint makes the optimization problem ill-behaved (the objective function could otherwise potentially increase without bound). The optimization problem can again be solved with simple gradient descent, starting from a random point in the space of feasible solutions and projecting to the space of feasible solutions at each step; projecting exactly onto both constraints is more complicated than the simple AM (Activation Maximization) with one norm constraint, but it follows from a straightforward algebraic computation.

Figure 5 illustrates this process for an optimization problem in 3 dimensions. We remind the reader that for simplicity this procedure is a sequence of gradient steps followed by projection to the constraints. Note that the projection operation always has two solutions (on the opposite sides of the feasible solutions circle/hypersphere, in our case) – we always pick the one that results in the highest activation value.

As discussed in the introduction to this section, when analyzing the directions of invariance, as given to us by the eigenvectors of the Hessian at the local maximum $\mathbf{x}_{opt}$, we did not observe any qualitatively interesting results. Our hypothesis is that there are many local directions—corresponding roughly to changing the background—and moving in those directions will not decrease the activation of the given unit[16]. Such an effect can also occur with our invariance manifold technique: the optimization procedure could conceivably move $\mathbf{x}^*_\varepsilon$ into directions that are of no interest to us (from a model analysis point of view)[17].

A way to counteract this effect is to move only in directions where there is variance in the data or, equivalently, dampen the directions in which there is no variance in the training data. More specifically, this can be accomplished by computing the whitening matrix W, via the zero-phase whitening (also called ZCA) transform (Bell and Sejnowski, 1997). This is the matrix which, when multiplied with $\mathbf{x} \in D_{train}$ spheres the data, i.e., $\mathrm{Cov}(y) = I$, where $\mathbf{y} = W\mathbf{x}$. Starting from $\mathbf{y}_{opt} = W\mathbf{x}_{opt}$, the search becomes:

$$\mathbf{y}^*_\varepsilon = \arg \max_{\mathbf{y}\ s.t.\ ||W^{-1}\mathbf{y}||=\rho \text{ and } ||W^{-1}(\mathbf{y}-\mathbf{y}_{opt})||=\varepsilon\rho} h_{ij}(\theta, W^{-1}\mathbf{y}) \tag{1}$$

[15] At $\varepsilon = 2$ the two (hyper-)spheres corresponding to the two constraints intersect at exactly one point. If ε is larger than 2, then the constraint cannot be satisfied anymore, since the spheres do not intersect (one is inside the other). See Figure 5.

[16] In other words, the learning procedure has managed to make units invariant to small background transformations.

[17] An interesting parallel can be made with an experiment that we performed, in which, instead of Activation Maximization we *minimize* the activation for each unit. The same "background effect" was observed. This suggests that the "activation landscape" of a hidden unit is similar to a ridge, in that there are a few directions of invariance—which are not easy to find—and quite a number of directions in which we can move and decrease the activation significantly.

That is, scale the directions in which we move by the amount of variance that the training data exhibits in those directions. Algorithm 1 contains the details of this procedure in pseudo-code format.

Algorithm 1 *Pseudo-code of the invariance computation procedure (eq. 1), using the whitening matrix to scale the directions in which we proceed. The* $\mathrm{projection}(\mathbf{y}_{new}, \mathrm{constraints}(\rho, \varepsilon, \mathbf{y}_{opt}))$ *operator signifies the function that projects* $\mathbf{y}_{new}$ *s.t.* $||W^{-1}\mathbf{y_{new}}|| = \rho$ and $||W^{-1}(\mathbf{y_{new}} - \mathbf{y}_{opt})|| = \varepsilon\rho$.

Require: $\mathbf{x}_{opt}$, W, and a learning rate μ

$\mathbf{y}_{opt} = W\mathbf{x}_{opt}$
$\mathbf{y}_{current} = \mathbf{y}_{opt}$
while not converged **do**
  $\mathbf{y}_{new} = \mathbf{y}_{current} + \mu \cdot \frac{\partial(h_{ij}(\theta, W^{-1}\cdot\mathbf{y}_{current}))}{\partial \mathbf{y}_{current}}$
  $\mathbf{y}_{current} = \mathrm{projection}(\mathbf{y}_{new}, \mathrm{constraints}(\rho, \varepsilon, \mathbf{y}_{opt}))$
end while
$\mathbf{y}^*_\varepsilon = \mathbf{y}_{current}$
return $\mathbf{y}^*_\varepsilon$

5.2 Results

We applied this method to a variant of the MNIST dataset, called *mnist-rot*, first presented by Larochelle et al. (2007). This is a dataset that contains rotated MNIST digits (random rotations, angle between $-\pi$ and π) and is being used in the community as a good check for empirically evaluating whether a given deep architecture is able to capture the rotational invariance in the data.

A sanity check for the invariance manifold technique just presented is to apply it to one of the 10 *output* units, corresponding to the predictions of the network for a given label. The hypothesis is that the results of the optimization technique on such units should be most interpretable (compared to other units in the network) and should be quite revealing of the invariances that are captured by the process of supervised learning.

Figure 6 (upper) presents several runs of AM on the output units corresponding to labels 4 and 5. A key observation in this case is that $\mathbf{x}_{opt}$ does not appear to be unimodal (as a function of random starting points). In fact, it would have been surprising otherwise: for instance, it is unlikely that the distribution of all rotated four-digits can be "captured" by a prototypical "four". Instead, we see a variety of rotated four- and five-digits.

Figure 6 (lower) contains an invariance manifold analysis: we picked the $\mathbf{x}_{opt}$ for the four-digit AM which had the highest activation value and then did four trials in which we varied the starting point of the optimization; this results in a *set of invariances* that characterize this unit. In fact, this was the only element of uncertainty in the optimization process—for a given ε we used the previous $\mathbf{x}^*_{\varepsilon-\delta}$ (meaning the solution with a slightly smaller ε) as the starting point. The startling observation is that even

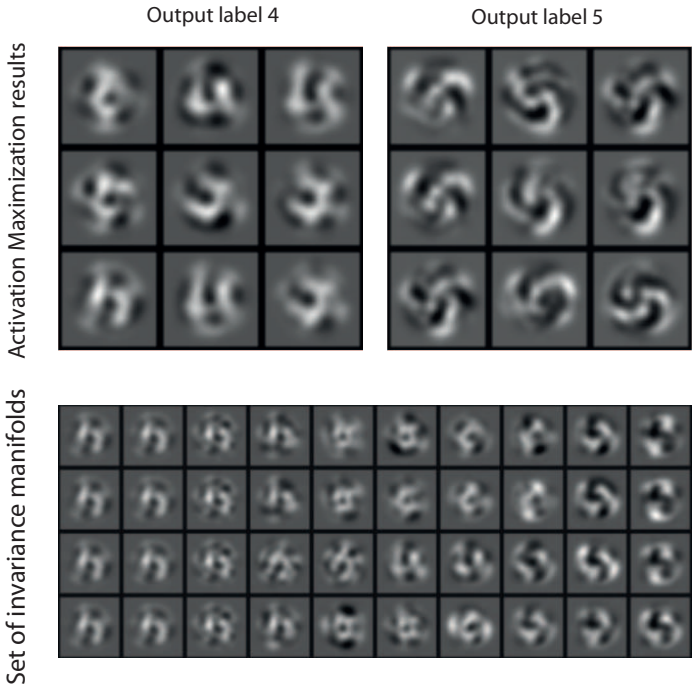

Figure 6: Upper*: output filter minima for the output units corresponding to digits 4 and 5 (upper).* Lower*: A* set *of invariance manifolds corresponding to digit 4, all starting from the same point (the best activation maximization result) and with a small random perturbation at the beginning of optimization; a row is one such trajectory / invariance manifold*

when only the very initial condition is changed, the invariance manifolds (from left to right on each row) become quite different. These manifolds also seem to interpretable, as they are capturing the various rotations that the output unit seem to be able to model.

5.3 Measuring invariance

Using the invariance manifold tool we can get an idea of the invariance for a given deep architecture model. Indeed, note that the activation value of a given unit j from a layer i, $h_{ij}(\mathbf{x}^*_\varepsilon)$ as one varies ε, can be considered as an indicator of invariance for a given unit: *the slower the unit's activation decreases as we increase ε the more invariant it is*. The intuition is the following: a unit whose activation drops down slowly has "carved" a manifold of the input space that is sufficiently large that even if we go far away from $\mathbf{x}_{opt}$ we can still maintain a high level of activation. Conversely, a unit whose activation drops down very fast has carved a small region of the space is therefore only responsible for only a few variations in the input data.

There is no established notion of a measure of invariance of a given unit in such a network. We argue however that, in a sense, our intuition can be used to reach a rather generic notion of invariance. Furthermore, to compute it, one does not need to specify a given *type* of invariance (though, as we shall see later in the discussion, this is also a limitation). This is in contrast with the work of Goodfellow et al. (2009), where the authors specify a series of input deformations (rotations, translations, etc) and an invariance measure that is computed for each unit.

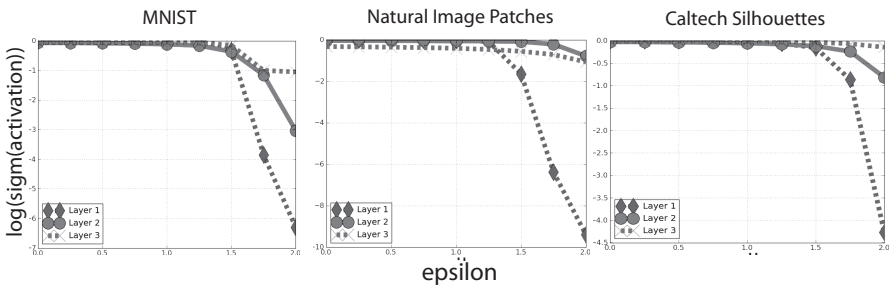

Figure 7: *Measuring the invariance of the units from different layers. From left to right, experiments on MNIST, Natural Image Patches and Caltech Silhouettes, with SDAE. The y-axis plots the sigmoid of the activation (in the log domain, for clarity) vs. the ε with which we move. The "*speed*" with which the curves decrease is what should be compared (layer 1 vs. layer 2 vs. layer 3).*

The main hypothesis that researchers in deep architectures have is that the upper layers of the models become more invariant to input transformations, presumably because of the increased level of abstraction represented by upper layers. Using our approach, this becomes a testable hypothesis: we simply need to compute the activation $h_{ij}(\mathbf{x}^*_\varepsilon)$ of each unit as ε increases, for all the units in a given layer. Figure 7 contains such an analysis, for MNIST, mnist-rot and Natural Image Patches. We observe that

in all cases the slope of the activation decrease (as ε increases) is *smaller for the first layer units compared to the second layer ones*; the second layer slopes for MNIST and Caltech Silhouettes are smaller than the third layer slopes as well. One could use this method to define a scalar measure of invariance, for instance from the area under the curve, which can then be used to compare models against each other. What the figure provides is new evidence to support the earlier observations of Goodfellow et al. (2009) that, in general, units from upper layers appear more invariant than those in the lower layer.

6 Conclusions and Future Work

We started from a simple desire: 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 (Dayan and Abbott, 2001, chapter 2.2), this approach has limitations, but we believe that such visualization techniques can help understand the nature of the functions learned by the network.

We describe three techniques for visualizing deep layers: activation maximization (AM) and sampling from an arbitrary unit are both new (to the best of our knowledge) and introduced in this work, while the linear combination technique had been previously introduced by Lee et al. (2008). We show the intuitive similarities between them and compared and contrasted them on three datasets. Our results confirm our intuitions about the hierarchical representations learned by deep architectures: namely that the higher layer units represent features that possess (meaningfully) more complicated structure and correspond to combinations of lower-layer features. The three techniques considered for visualization give rise to meaningfully different results: as posited in the introduction, we found that a sampling-based method produces a distribution of training-set-like samples, which may require further processing to make sense of what specifically the chosen units captures. Conversely, AM (and, to a lesser extent, the linear combination method) make it possible to get a "part"-like representation of each unit, an arguably more interpretable representation.

We also find that the two deep architectures considered learn quite different features. An unexpected result (Figure 2) is the discovery that, for natural image patches, uninformative-looking first-layer filters of a Deep Belief Network do not necessarily tell the whole story: we show that second-layer units can model edge detectors and grating filters in the same model. The implication of this result is that higher-layer units can be an important tool for comparing models and provides a justification for seeking to understand and visualize what the upper-layer units in a deep architecture do; such a result should be interpreted in the context of the standard approach used in many papers on deep architectures (Osindero and Hinton, 2008; Larochelle et al., 2009), which is that of simply looking at first-layer filters, in addition to test error performance.

Further leveraging the AM methodology, we turn to the question of exploring the invariances that are learned by individual units in a network. Again, we can cast this question as an optimization problem. We explore this in detail for the output units of a

supervised network trained on rotated digits. These explorations confirm our intuitions that these manifolds essentially capture the kinds of general invariances present in the data and learned by the model. Finally, this investigation naturally provides a way to visualize and measure invariance. This experiment allowed us to compare layers in a fairly generic way (with respect to how "invariant" the average activation of a unit is, as we move away from the result of AM), without actually specifying the set of invariances by hand, or generating data in any way; as with AM, this invariance analysis is applicable to a large class of deep architectures.

The same procedures (AM and invariance analyses) 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. We have already performed a basic analysis along these lines—in Erhan et al. (2010), Figures 3 and 4, where we show the influence of pre-training on a deep network. However, we feel that more work is needed in order to better understand the qualitative effect of pre-training for supervised learning and visualization/invariance analysis tools could be helpful in this respect.

We would be interested in comparing with Goodfellow et al. (2009)'s approach of hand-crafted input transformations (such as translations, rotations etc.), and the measurements of invariance of upper-layer units as a function of these transformations. Our belief is that analysis methods that rely on specific invariances are limited in the story they can tell us, because we would like to measure invariance to variations that are not known a priori. The method we presented in this paper is generic with respect to the input transformations and is thus a generic way of measuring invariance; in this sense, it is an interesting alternative to Goodfellow et al. (2009)'s approach. Nonetheless, one could reasonably question the interpretability of the invariance manifolds that our method uncovers. Would it be possible to project or decompose the manifold of a given unit to a set of known invariances? Could we group the units of the layer according to certain types of invariance?

Future research will concentrate on exploring such questions. Ideally, a future method for analysis would be able to detail, for a given unit, the level of invariance with respect to (for example) rotation, translation and scaling of the input data and provide us with an idea of how invariant it is to other transformations of the input that are not in the list. Our work is a step in such a direction. The analysis in Figure 5 could be extended such that the search space of the invariance manifold is limited to inputs corresponding to only rotations (or only translations or etc.) of $\mathbf{x}_j^*$, the AM output; by computing curves such as the ones in Figure 7, for each such transformation separately, one could then come up with at least a *relative* notion of invariance, meaning that we could understand whether a unit is more invariant to rotations or to translations. From there, we could compare entire layers or model instances, and we might also be able to compare the behaviour of higher level units in a deep network to features and invariances that are presumed to be encoded by the higher levels of the visual cortex Lee et al. (2008).

Acknowledgments

We would like to thank Pascal Vincent and Ian Goodfellow for helpful discussions.

References

Bell, A. and Sejnowski, T. J. (1997). The independent components of natural scenes are edge filters. *Vision Research*, 37:3327–3338.

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

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160. MIT Press.

Berkes, P. and Wiskott, L. (2002). Applying slow feature analysis to image sequences yields a rich repertoire of complex cell properties. In Dorronsoro, J. R., editor, *Proc. Intl. Conf. on Artificial Neural Networks - ICANN'02*, Lecture Notes in Computer Science, pages 81–86. Springer.

Berkes, P. and Wiskott, L. (2006). On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields. *Neural computation*, 18(8):1868–1895.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience*. The MIT Press.

Desjardins, G., Courville, A., and Bengio, Y. (2010). Parallel tempering for training of restricted Boltzmann machine. In *Proceedings of AISTATS 2010*, volume 9, pages 145–152.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660.

Fei-Fei, L., Fergus, R., and Perona, P. (2004). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Computer Vision and Pattern Recognition Workshop, 2004 Conference on*, page 178.

Goodfellow, I., Le, Q., Saxe, A., and Ng, A. (2009). Measuring invariances in deep networks. In Bengio, Y., Schuurmans, D., Williams, C., Lafferty, J., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 646–654.

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

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

Hinton, G. E., Osindero, S., Welling, M., and Teh, Y. (2006b). Unsupervised discovery of non-linear structure using contrastive backpropagation. *Cognitive Science*, 30(4).

Larochelle, H., Bengio, Y., Louradour, J., and 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., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Ghahramani, Z., editor, *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, pages 473–480. ACM.

Lee, H., Ekanadham, C., and Ng, A. (2008). Sparse deep belief net model for visual area V2. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 873–880. MIT Press, Cambridge, MA.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Bottou, L. and Littman, M., editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, Montreal (Qc), Canada.

Loosli, G., Canu, S., and Bottou, L. (2007). Training invariant support vector machines using selective sampling. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA.

Marlin, B., Swersky, K., Chen, B., and de Freitas, N. (2009). Inductive principles for restricted boltzmann machine learning. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, volume 9, pages 509–516.

Olshausen, B. A. and 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. and Hinton, G. E. (2008). Modeling image patches with a directed hierarchy of markov random field. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1121–1128, Cambridge, MA. MIT Press.

Ranzato, M., Boureau, Y.-L., and LeCun, Y. (2008). Sparse feature learning for deep belief networks. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1185–1192, Cambridge, MA. MIT Press.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 1137–1144. MIT Press.

Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. In Bottou, L. and Littman, M., editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, pages 1033–1040. ACM.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM.

Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, New York, NY, USA. ACM.