How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation

Yoshua Bengio
Université de Montréal
CIFAR Fellow

Abstract

In this paper we propose to exploit *reconstruction* as a layer-local training signal for deep learning, be it generative or discriminant, single or multi-modal, supervised, semi-supervised or unsupervised, feedforward or recurrent. Reconstructions can be propagated in a form of target propagation playing a role similar to back-propagation but helping to reduce the reliance on back-propagation in order to perform credit assignment across many levels of possibly strong non-linearities (which is difficult for back-propagation). A regularized auto-encoder tends produce a reconstruction that is a more likely version of its input, i.e., a small move in the direction of higher likelihood. By generalizing gradients, target propagation may also allow to train deep networks with discrete hidden units. If the auto-encoder takes both a representation of input and target (or of any side information) in input, then its reconstruction of input representation provides a target towards a representation that is more likely, conditioned on all the side information. A deep auto-encoder decoding path generalizes gradient propagation in a learned way that can thus handle not just infinitesimal changes but larger, discrete changes, hopefully allowing credit assignment through a long chain of non-linear operations. For this to work, each layer must be a good denoising or regularized auto-encoder itself. In addition to each layer being a good auto-encoder, the encoder also learns to please the upper layers by transforming the data into a space where it is easier to model by them, flattening manifolds and disentangling factors. The motivations and theoretical justifications for this approach are laid down in this paper, along with conjectures that will have to be verified either mathematically or experimentally.

1 Introduction

Deep learning is an aspect of machine learning that regards the question of learning multiple levels of representation, associated with different levels of abstraction (Bengio, 2009). These representations are distributed (Hinton, 1989), meaning that at each level there are many variables or features, which together can take a very large number of configurations. Deep representations can be learned in a purely unsupervised way (sometimes with a generative procedure associated with the model), in a purely supervised way (e.g., a deep feedforward network), or in a semi-supervised way, e.g., with unsupervised pre-training (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007). It is possible to build deep representations that capture the relationships between multiple modalities (Srivastava and Salakhutdinov, 2014) and to model sequential structure through recursive application (Bottou, 2011) of learned representation-to-representation transformations, e.g. with recurrent neural networks (Sutskever, 2012) or recursive neural networks (Socher *et al.*, 2011).

Deep learning methods have essentially relied on three approaches to propagate training signals across the different levels of representation:

1. Greedy layer-wise pre-training (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007): the lower levels are trained without being influenced by the upper levels, each only trying to find a better representation of the data as they see it from the output of the previous level of representation. Although this approach has been found very useful as an initialization for the second or third approach, its potential disadvantage is that by itself it does not provide a global coordination between the different levels.

2. Back-propagated gradients: the features must be continuous and gradients with respect to lower layers (through many non-linearities) appear less able to correctly train them, especially when considering deeper networks trying to learn more abstract concepts obtained from the composition of simpler ones (Bengio, 2009; Gulcehre and Bengio, 2013). This is especially true of very deep or recurrent nets (Hochreiter, 1991; Bengio *et al.*, 1994; Hochreiter and Schmidhuber, 1997; Pascanu *et al.*, 2013), involving the composition of many non-linear operations. Back-propagation and gradient-based optimization can be used in a supervised setting (e.g., for classification or regression) or an unsupervised setting, e.g., for training shallow (Vincent *et al.*, 2008) or deep (Martens, 2010) auto-encoders and deep generative models (Goodfellow *et al.*, 2014).

3. Monte-Carlo Markov chains (MCMC): the stochastic relaxation performed in models such as Markov random fields and Boltzmann machines (including the Restricted Boltzmann Machines) propagates information about the observed variables into the parameters governing the behavior of latent variables (hidden units), so as to make the sufficient statistics of configurations generated by the model as close as possible as to those obtained when the observed visibles are clamped. Unfortunately, with some form of MCMC in the inner loop of training, mixing difficulties may limit our ability to learn models that assign probability in a sharp way, near manifolds (modes) that are far from each other (Bengio *et al.*, 2013a; Bengio, 2013). This becomes especially troublesome when one tries to learn models of complex distributions involving a manifold structure in high dimension.

There is a fourth and insufficiently explored approach, to which the proposal discussed here belongs, based on *target propagation* (LeCun, 1986). Back-propagation and target propagation are identical when the target is viewed as an infinitesimal direction of change and the gradient can be computed analytically (LeCun, 1987). However, target propagation can also be potentially applied to the case where the representation involves discrete values (LeCun, 1986). Target propagation was previously also proposed (Bengio *et al.*, 1994) as a way to defeat some of the optimization difficulties with training recurrent neural networks, due to long-term dependencies in sequential data. This target propagation viewpoint is also related to optimization approaches where the representation values (i.e., hidden unit activations) are free variables that can be optimized over (Carreira-Perpinan and Wang, 2014). In this paper, we propose to use auto-encoders (or conditional auto-encoders, whose "code" can depend on side information as well) to provide learned reconstructions that can be used as targets for intermediate layers. In a sense, we are proposing to *learn the back-propagation computation*. In doing so, we generalize back-propagation (for example allowing one to handle discrete-valued elements of the representation) and make it more biologically plausible as a mechanisms for brains to perform credit assignment through many levels of a deep computation, including a temporally recurrent one. A recent paper proposes the reweighed wake-sleep algorithm, that learns a deep generative model and can also handle discrete latent variables (Bornschein and Bengio, 2014), while being based on a generalization of the wake-sleep algorithm (Hinton *et al.*, 1995). The proposed target propagation indeed has some similarity to both the original wake-sleep algorithm and the reweighted wake-sleep algorithm. Unlike the latter, it does not require multiple inference samples in order to obtain a reasonable gradient during training.

This paper starts by what is probably the simplest and most natural context for the idea of using reconstruction for target propagation: that of generative models in which each layer is trained as an auto-encoder, and at any level, the representation h has a distribution prior $P(h)$ which is implicitly captured by the upper levels. The mathematical framework we propose is based on recent work on variational auto-encoders (Kingma and Welling, 2014; Rezende *et al.*, 2014), i.e., a lower bound on the log-likelihood, but can also be interpreted as trying to match the joint distribution of latent and observed variables under the generative model and under the composition of the data generating distribution with a "recognition network" that predicts latent variables given inputs. The main idea is that upper auto-encoders provide by their reconstruction a proposed change which indicates the

direction of higher prior probability, and can be used as a proxy for the gradient towards making the lower levels produce outputs that are more probable under the implicit model of the upper levels.

This paper then extends this framework to the classical supervised setting in which both an input variable x and a target or output variable y are involved. In this case, when y is observed, it constrains the reconstruction of a layer h that was initially computed based only on x, i.e., the reconstruction provides a representation value that is compatible with both x and y. This approach can be naturally generalized to multi-modal data, where instead of x and y, one can think of different sensory modalities $x^{(i)}$. Again, the reconstruction of a representation $h^{(i)}$ which initially only depends on $x^{(i)}$ is made to depend on both $x^{(i)}$ and the observations made with other modalities $x^{(j)}$. Finally, this framework is applied in the very interesting context of recurrent networks, in which we have a sequence of x_t's, and the reconstruction of a representation h_t of the past sequence incorporates the constraints induced by future observations.

2 Stacked Auto-Encoders as Generative Models

2.1 Preliminaries and Relevant Recent Work

2.1.1 Notation

Denote h_l the layer-l representation, which we generally think of as a random variable, and $h = (h_1, h_2, \ldots)$ all the layers of a generative model. Denote $p(x, h)$ the joint distribution over x and h, structured as a directed graphical model $h_L \Rightarrow \ldots h_2 \Rightarrow h_1 \Rightarrow x$ with a *chain structure*:

$$P(x, h) = P(x|h_1)P(h_1|h_2)\ldots P(h_{L-1}|h_L)P(h_L). \tag{1}$$

With $x = h_0$, one can view each $P(h_l|h_{l+1})$ as one layer of a generative network, or *decoder* network, that maps top-level representations into low-level samples.

This graphical structure is the same as in sigmoidal belief networks (Neal, 1992), but just like for Helmholtz machines (Dayan *et al.*, 1995), we will also consider a *recognition* network, or *encoder*, or approximate inference network, which computes in the reverse direction, starting with the unknown data generating distribution, which we denote $Q(X)$:

$$Q(x, h) = Q(h_L|h_{L-1})\ldots Q(h_2|h_1)Q(h_1|x)Q(x). \tag{2}$$

Accordingly, we define $P(h_l)$ and $Q(h_l)$ is the marginals respectively associated with the joints P and Q over all the variables.

2.1.2 Motivating Deterministic Latent Variables

Most deep generative models proposed in the past, such as deep Belief networks (Hinton *et al.*, 2006), Helmholtz machines (Dayan *et al.*, 1995) and deep Boltzmann machines (Salakhutdinov and Hinton, 2009), have the property that lots of "noise" gets injected at every level of the multilayer generative model. At the lowest level ($P(x|h_1)$), it typically means that the generated x is the "addition" of an expected $E[x|h_1]$ and some iid "noise". This iid noise shows up in generated samples as high-frequency spatial (for images) or temporal (for acoustics) noise which is not at all similar to what is observed in the training data. It necessarily moves away from any low-dimensional manifold near which the distribution concentrates, ruining the possibility of being able to capture such a sharp distribution.

A similar remark can be made regarding the noise injected when sampling h_1 given h_2, because the mapping from h_1 to x is generally simple (e.g. affine + simple saturating non-linearity). The effect of such noise is especially striking when h_1 is a vector of stochastic binary units: when h_1 is sampled from $P(h_1|h_2)$, many independent coin flips are generated to choose the different values h_{1i}. These independent sources of noise must then somehow be transformed into well-formed x through a simple mapping (such as the affine transformation composed with sigmoidal non-linearity typically used in such models). This could only happen if both $P(x|h_1)$ and $P(h_1|h_2)$ are almost deterministic (i.e. no coin flips), or if the dimension of h_1 is so large that the independent noise sources cancel each other. Note that if $P(x|h_1)$ and $P(h_1|h_2)$ are nearly deterministic (i.e., nearly diracs), then the learning procedures typically proposed for such models break down. For example,

when the weights of a Boltzmann machine become large, MCMCs do not mix and training stalls. In the case of the recently proposed variational auto-encoder (Kingma and Welling, 2014; Rezende *et al.*, 2014) variants (which is based on essentially the same criterion that is exploited here but has been done with stochastic encoders), the stochastic gradients would explode as the variance component of some units would approach 0. See the discussion in Section 2.2.2 below to clarify that mathematically.

Another, very different and very interesting view of a deep generative model is offered in the recent work on adversarial generative networks (Goodfellow *et al.*, 2014). In that case, randomness can be viewed as injected at the top level (possibly at lower layers too, but that is a choice of the designer), while the intermediate levels are considered to be purely deterministic and continuous transformations. What could make learning in such networks difficult is that one still has to assign credit (by back-prop) to all the layers of a deep net. Another less well understood potential issue is that for each update step of the generator network, a discriminator network must be sufficiently re-optimized to continuously track the decision surface between examples generated by the model and training examples. Finally, another potential issue (which can be seen as part of the optimization problem) is that if the generator concentrates on some modes of the data distribution, there is very little pressure to make it generate samples for the other modes (only for the rare generated samples that approach these other modes does the generator get a signal). However, a great innovation of the adversarial network is that it opens the door to generative models in which the noise is injected at the top level, which we consider here to be a very important feature.

In any case, the basic idea in many of these models is that although the top-level prior $P(h_L)$ is going to be simple, e.g., factorial or a single RBM, the lower levels gradually transform $P(h_L)$ into more complex distributions $P(h_l)$, ending in $P(x)$. For example, with the manifold learning view, we can imagine $P(h_L)$ as essentially uniform on one or several manifolds where the distribution concentrates, with h_L representing a coordinate system for the data in an abstract space where all the variables are independent. That flat manifold is then distorted (and possibly broken down into separate sub-manifolds for different classes) in complicated non-linear ways. In this view, the job of the generative network is really just to transform and distort the space such that a simple distribution gets mapped into a good approximation of the data generating distribution. Similarly, the job of the encoder networks (the $Q(h|x)$) is to map a complicated distribution into a simpler one, layer by layer, to map a highly curved manifold into a flat one. Under that manifold-learning perspective, we want most of the "noise" to be injected high in the hierarchy. That "noise" represents the high-level choices about the content of the generated x. For example, the top-level factors in a deep net modeling images of a single object might include not just the object category but all of its attributes, geometrical pose parameters, lighting effects, etc. We know from the physics of image synthesis that the mapping from such high-level variables to pixels is highly non-linear, which means that these factors should be chosen (i.e. "sampled") high up in the deep network.

This discussion motivates the use of at least some units (maybe all, or the majority) at each level that are deterministic functions (in the generative network) of the higher level units. This is a significant design choice because many current learning algorithms for deep generative nets are not numerically well suited for learning such deterministic latent variables. As mentionned above, when latent variables become nearly deterministic, MCMC methods tend to break down. Using stochastic EM (Tang and Salakhutdinov, 2013) importance sampling, as in Bornschein and Bengio (2014), or the current formulations of variational auto-encoders (Kingma and Welling, 2014; Rezende *et al.*, 2014) may also become problematic when the distributions considered become nearly deterministic.

This paper is starting from this observation and asking how we can train these kinds of directed models when the latent variables are, at least in part (or on some large fraction of examples), nearly deterministic functions of the input.

2.2 A Generative Stack of Auto-Encoders

2.2.1 Matching Recognition and Generative Networks

We propose here that an appropriate objective for training a deep encoder/decoder pair as introduced above is that the joint distribution over h and x generated by P matches the joint distribution generated by Q. Later, we will see that this can be reduced to having the marginal distributions $P(H_i)$ and

$Q(H_i)$ match each other, when the layerwise auto-encoder pairs are good in terms of minimizing reconstruction error.

The motivation is straightforward: if $P(X, H)$ matches well $Q(X, H)$, then it necessarily means that its marginals also match well, i.e., the generative distribution $P(X)$ matches well the data generating distribution $Q(X)$. This criterion is thus an alternative to maximum likelihood (which tries to directly match $P(X)$ to $Q(X)$), with the potential advantage of avoiding the associated intractabilities which arise when latent variables are introduced.

Mathematically, this objective of matching $P(X, H)$ and $Q(X, H)$ can be embodied by the KL-divergence between Q and P, taking Q as the reference, since we want to make sure that P puts probability mass everywhere that Q does (and especially where $Q(X)$ does).

The criterion $KL(Q||P)$ can be decomposed in order to better understand it:

$$\begin{aligned} KL(Q||P) =& E_{(x,h)\sim Q(X,H)} \log \frac{Q(x)Q(h|x)}{P(x|h)P(h)} \\ =& - H(Q) - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log P(x|h) - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log P(h) \end{aligned} \quad (3)$$

We distinguish three terms:

1. The **entropy** of the joint distribution of h and x under Q. Because $Q(X)$ is fixed, this turns out to be equivalent to the average entropy of $Q(H|x)$, for $x \sim Q(X)$:
$$\begin{aligned} H(Q) =& - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \left(\log Q(x) + \log Q(h|x)\right) \\ =& - E_{x\sim Q(X)} \left(\log Q(x) + E_{h\sim Q(H|x)} \log Q(h|x)\right) \\ =& H(Q(X)) + E_{x\sim Q(X)} H(Q(H|x)) \end{aligned} \quad (4)$$
where $H(P(A|b))$ is the entropy of the conditional distribution of A, given $B = b$. This allows us to rewrite the KL criterion as follows:
$$\begin{aligned} KL(Q||P) =& - H(Q(X)) - E_{x\sim Q(X)} H(Q(H|x)) - \\ & E_{(x,h)\sim Q(X,H)} \log P(x|h) - E_{h\sim Q(H)} \log P(h) \end{aligned} \quad (5)$$
2. The **match of observed $Q(H)$ to the generative model $P(H)$**, measured by the log-likelihood of the samples $h \sim Q(H|x), x \sim Q(X)$ according to the prior $P(h)$. Since both $P(h)$ and $Q(h|x)$ are free, we see that $P(h)$ will try to match the samples coming from the encoder, but *also* that $Q(h|x)$ will try to put h in places where $P(h)$ is high.
3. The **reconstruction log-likelihood** $\log P(x|h)$, when $h \sim Q(H|x)$. This is the *traditional criterion used in auto-encoders.*

Note that this criterion is equivalent to the training criterion proposed for the variational auto-encoder (Kingma and Welling, 2014; Rezende *et al.*, 2014), i.e., it can also be justified as a variational bound on the log-likelihood $E_{x\sim Q(X)} \log P(x)$, where the bound is tight when $Q(h|x) = P(h|x)$, and the bound arises simply out of Jensen's inequality or explicitly pulling out $KL(Q(H|x)||P(H|x))$:

$$\begin{aligned} -E_{x\sim Q(X)} \log P(x) =& - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log P(x) \\ =& - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log \frac{P(x,h)Q(h|x)}{P(h|x)Q(h|x)} \\ =& - E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log \frac{Q(h|x)}{P(h|x)} + \log \frac{P(x,h)}{Q(h|x)} \\ =& E_{x\sim Q(X)} KL(Q(H|x)||P(H|x)) + E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log \frac{Q(h|x)}{P(x,h)} \\ \geq& E_{x\sim Q(X)} E_{h\sim Q(H|x)} \log \frac{Q(h|x)}{P(x,h)} \\ =& KL(Q||P) + H(Q(X)) \end{aligned} \quad (6)$$

where the last line comes from inspection of the previous line compared with Eq. 5, i.e., we have the (fixed) entropy of the data generating distribution plus the overall $KL(Q||P)$ that we considered here as a training criterion. In the penultimate line we recognize the variational auto-encoder training criterion (Kingma and Welling, 2014; Rezende *et al.*, 2014).

2.2.2 Why Deterministic Encoders

With the above $KL(Q||P)$ criterion, which is, as shown above, the same (up to a non-trained constant $H(Q(X))$) as the variational auto-encoder (Kingma and Welling, 2014; Rezende *et al.*, 2014), some numerical issues arise as the learner tries to discover some parts of the representation which would have small noise, i.e., for which $Q(H|x)$ has small entropy, at least for some components of it. We develop that below.

Consider $Q(H|x)$ a Gaussian with a mean $\tilde{f}(x)$ and a diagonal variance $\sigma(x)$ (with one value for each component of h). Even though the gradient of the entropy term with respect to $\tilde{f}(x)$ vanishes in expectation (as discussed above), the stochastic gradient of the entropy term can be arbitrarily large in magnitude as $\sigma_i(x)$ approaches 0 (for any i). More generally, the same problem will occur if the covariance matrix is not diagonal, so long as some eigenvector of the covariance matrix goes to 0. For example, in the scalar case, the stochastic gradient (for a given sample of $h \sim Q(H|x)$) would be proportional to $\frac{\tilde{f}(x)-h}{\sigma^2(x)}$, which goes to infinity when $\sigma(x)$ becomes small, *even though in average it could be 0* (as is the case of the gradient due to the entropy term). Note however that this gradient may be integrated out over $Q(H|x)$ in the Gaussian case, but this does not completely eliminate the problem, as the gradient with respect to $\sigma^2(x)$ is still proportional to $1/\sigma^2(x)$.

In other words, even though the criterion allows in principle to learn nearly deterministic transformations of x, for numerical reasons the gradients become badly behaved when some units try to become deterministic (i.e., when their $\sigma_i(x)$ becomes small). This in turn prevents learning nearly deterministic components of the representation. The consequence is that $P(x|h)$ is necessarily going to have high entropy, because too much crucial information about x has been lost in $h \sim Q(H|x)$. This entropy necessarily corresponds to "adding noise" at the level of x, and generally this is independent noise, which shows up as ugly uncorrelated bits in the generated signals. The only way to avoid that would be to make $P(x|h)$ a highly multimodal complicated distribution (and this is the route that Ozair *et al.* (2014) have taken). Therefore, if we want to keep $P(x|h)$ unimodal (with a simple partition function), it is hypothesized here that we are better off forcing some (or even all) outputs of the encoder (which will carry the "signal") to be deterministic functions of x.

2.2.3 A Criterion for Each Layer

Let us consider each layer h_l of the chain structure (Eq. 1) and apply the above $KL(Q||P)$ decomposition as if we were only considering h_l as the latent variable. The objective of this approach is to provide a training signal for each layer h_l and for the parameters of the associated encoders and decoders (the ones encoding into h_l and reconstructing lower layers and the ones encoding and reconstructing h_l from above). We will consider the upper layers h_{l+1}, h_{l+2}, etc. as implicitly providing a prior $P(h_l)$ since they will be trained to implicitly model $h_l \sim Q(H_l)$ through the samples h_l that they see as training data. Hence the criterion for layer h_l is:

$$KL(Q^l||P) = -H(Q(X)) - E_{x\sim Q(X)}H(Q(H_l|x)) - E_{(x,h_l)\sim Q(X,H_l)}\left[\log P(x|h_l) + \log P(h_l)\right]. \tag{7}$$

Let us consider each of these terms in turn.

1. We ignore the first term because it is fixed.
2. The second term is the average entropy of the conditional distribution $Q(H_l|x)$. If we choose $Q(H_l|x)$ to be noise-free, i.e., the deep encoder $\tilde{f}_l$ that maps x to h_l is not stochastic and it produces
$$h_l = \tilde{f}_l(x) \tag{8}$$
with probability 1, then the entropy of $Q(H_l|x)$ is zero and cannot change, by this design choice.
3. The third term is a reconstruction negative log-likelihood. If we assume that $Q(X|h_l)$ is nearly deterministic, then it is captured by a decoder function $\tilde{g}_l$ that maps h_l to x. Then the reconstruction log-likelihood is minimized totally if $\tilde{f}_l$ and $\tilde{g}_l$ form what we call a *perfect auto-encoder relative to* $Q(X)$. More precisely, if h_l is computed deterministically as in Eq. 8, with $x \sim Q(X)$, then the third term is totally minimized so long as
$$\tilde{g}_l(\tilde{f}_l(x)) = x \tag{9}$$

when $x \sim Q(X)$. Note that the auto-encoder $\tilde{g}_l \circ \tilde{f}_l$ will be a perfect auto-encoder if each of the layer-wise auto-encoders below level l is also a perfect auto-encoder. Let us denote each layer-wise auto-encoder pair by the encoder f_l and the decoder g_l, and accordingly define

$$\tilde{f}_l(x) = f_l(f_{l-1}(\dots f_2(f_1(x)))) \tag{10}$$

and

$$\tilde{g}_l(h_l) = g_1(g_2(\dots g_{l-1}(g_l(h_l)))). \tag{11}$$

Then we have that

$$g_i(f_i(h_{i-1})) = h_{i-1}, \ \ \forall h_{i-1} \sim Q(H_{i-1}), \ \forall i \quad \Rightarrow \quad \tilde{g}_i(\tilde{f}_i(x)) = x, \ \ \forall x \sim Q(X), \ \forall i \tag{12}$$

Hence it is enough, to cancel the third term, to make sure that each layer l is a perfect auto-encoder for samples from its "input" $h_l \sim Q(H_l)$. Note that there is an infinite number of invertible mappings f and g, so up to now we have not specified something really "interesting" for the system to learn, besides this ability to auto-encode data at each layer.

4. The last term is the most interesting one, i.e., matching the marginals $P(H_i)$ and $Q(H_i)$. At any given layer l, it is doing two things, if we consider the pressure on Q and the pressure on P separately:

   (a) This marginal matching term is asking the upper layers to learn a prior $P(h_l)$ that gives high probability to the samples $h_l \sim Q(H_l)$. This just means that the "input data" $h_l \sim Q(H_l)$ seen by the upper layers must be well modeled by them. For example, the upper layers prior could be represented by a deep denoising auto-encoder trained with $h_l \sim Q(H_l)$ as data. In that case we want h_l to be the target for updating the reconstruction of the auto-encoder. The input of that auto-encoder could be h_l or a corrupted version of it.

   (b) The marginal matching term is also asking the lower layers to choose an invertible encoding $\tilde{f}_l$ such that the transformation of $Q(X)$ into $Q(H_l)$ yields samples that have a high probability under the prior $P(H_l)$. In other words, it is *asking the lower layers to transform the data distribution (which may be very complicated, with many twists and turns) into an easy to model distribution (one that the upper layers can capture)*. Note however how the pressure on $Q(H_l)$ (i.e., on $\tilde{f}_l$ and on $P(H_l)$ are asymmetric. Whereas $P(H_l)$ is simply trained to model the "data" $h_l \sim Q(H_l)$ that it sees, $\tilde{f}_l$ is pressured into producing samples more towards the modes of $P(H_l)$, thus tending to make $Q(H_l)$ more concentrated and $\tilde{f}_l$ contractive. Overall there are thus two forces at play on $\tilde{f}_l$: to minimize the input space reconstruction error, it would like to spread out all of the x training examples as uniformly as possible across h-space. But an opposing force is at play, making $Q(H_l)$ concentrate in a few smaller regions (the modes of $P(H_l)$). If it weren't for the reconstruction error, $\tilde{f}_l$ would just map every training example to one or a few modes of $P(H_l)$ and $P(H_l)$ would just become highly concentrated on those modes. But that would make reconstruction error of the input very high. So the compromise that seems natural is that $\tilde{f}_l$ contracts (towards modes of $P(H_l)$) in the local directions that it does not need to represent because they do not correspond to variations present in the data (i.e., it contracts in the directions orthogonal to the manifolds near which $Q(X)$ concentrates). However, it yields a more uniform distribution in the directions of variation of the data, i.e., on the manifolds. Interestingly, we can view the pressure from $P(H_l)$ onto the encoder as a *regularizer* that prevents the auto-encoder from just learning a general-purpose identity function (invertible everywhere). Instead, it is forced to become invertible only where $Q(X)$ is non-negligeable.

   Below we discuss how a training signal for the lower-levels encoder $\tilde{f}_l$ could be provided by the upper auto-encoder that captures $P(h_l)$.

Note that if we include $h_0 = x$ as one of the layers on which the above criterion is applied, we see that this is just a proxy for maximum likelihood: at level 0, the only term that remains "trainable" is $E_{x \sim Q(X)} \log P(X)$. If $P(X)$ is estimated by a deep regularized auto-encoder, then this proxy is the usual training criterion for a regularized auto-encoder. One reason why we believe that the criteria

for the other layers helps that it provides a training signal for every intermediate layers of the deep auto-encoder, thus hopefully making the optimization easier. It also justifies the top-down directed generative procedure associated with Eq. 1, which is not obviously applicable to an arbitrary deep auto-encoder, as well as the MAP inference procedure discussed in Section 8.4.

2.2.4 How to Estimate a Target

In the last step of the above decomposition, we are required to specify a change of $\tilde{f}_l$ such as to move $h_l = \tilde{f}_l(x)$ to a nearby value $\hat{h}_l$ that has a higher probability under $P(h_l)$ (or return $\hat{h}_l = h_l$ if h_l is already a mode).

Fortunately, we can take advantage of a previously proven theoretical result (Alain and Bengio, 2013), which has been shown in the case where h_l is continuous and the training criterion for the auto-encoder is simply squared error. If a very small quantity of noise is injected in the auto-encoder, then the difference between the reconstruction $\hat{h}_l$ and the (uncorrupted) input h_l of the auto-encoder is proportional to an estimator of $\frac{\partial \log Q(h_l)}{\partial h_l}$, where $Q(h_l)$ here denotes the "true" distribution of the data seen by the auto-encoder (i.e. samples from $Q(H_l)$).

The basic reason why a denoising auto-encoder learns to map its input to a nearby more probable configuration is that it is trained to do so: it is trained with (input,target) pairs in which the input is a corrupted version of the target, and the target is a training example. By definition, training examples are supposed to have high probability in average (this is what maximum likelihood is trying to achieve), and a random move around a high probability configuration is very likely to be a low probability configuration. In the maximum likelihood setup, we are trying to force the model to put a high probability on the examples and a small probability everywhere else. This is also what is going on here: we are telling the auto-encoder that it should map points that are not training examples to a nearby training example. This mental picture also highlights two things: (1) the auto-encoder training criterion is more local than maximum likelihood (if the noise level is small, it only sees configurations in the neighborhood of the data), but higher noise levels should mitigate that, and (2) if we increase the noise level we will make the model do a better job at killing off spurious modes (configurations that are probable under the model but should not), however the model then might start fuzzying its reconstruction because the same corrupted configuration could then be reached from many training examples, so the reconstruction would tend to be somewhere in the middle, thus filling the convex between neighboring training examples with high probability values. The latter observation suggests the nearest-neighbor training procedure sketched in Section 2.2.8. Another (orthogonal) solution to spurious modes that was previously proposed (Bengio *et al.*, 2013b) and that works well is the *walkback procedure* in which the noise is not completely random but along paths following reconstruction, i.e., we let the learner go from a training example towards a spurious mode by iterative encode/decode steps and then punish it by telling to reconstruct the starting training example (this is similar to Contrastive Divergence).

Although we can consider $\hat{h}_l - h_l$ as a proxy for $\frac{\partial \log P(h_l)}{\partial h_l}$ and providing a vector field (a vector for each point h_l), keep in mind that this vector is not guaranteed to be a proper gradient, in the sense that integrating it through different paths could give slightly different results. Only in the asymptotic non-parametric limit is $\hat{h}_l - h_l$ converging to a gradient field. The previous analysis of denoising auto-encoders (Alain and Bengio, 2013; Bengio *et al.*, 2013b) clearly shows that the auto-encoder implicitly estimates a distribution $P(h_l)$ so as to match it as well as possible to its training distribution $Q(H_l)$. This is consistent with even earlier results showing that the denoising criterion is a regularized form of score matching (Vincent, 2011) called denoising score matching (Swersky *et al.*, 2011). It is also consistent with a geometric intuition explained in Bengio *et al.* (2013c) suggesting that if the auto-encoder is regularized (in the sense that it cannot perfectly reconstruct every possible input), then the output of the encoder will be most (or even only) locally sensitive to the locally probable variations of the data around the input, and only reconstruct well for inputs that are along these highly probable regions (manifolds). Indeed, the auto-encoder only has limited representation resources (this is due to the regularization, contraction of the encoder and decoder functions) and in order to minimize reconstruction error it must use this capacity where it is really needed, i.e., to capture the variations in regions of high density of the data distribution, while ignoring variations not present in the data, by mapping unlikely input configurations towards nearby more likely configurations.

Based on this intuition, we conjecture that the above result regarding the meaning of $\hat{h}_l - h_l$ can be generalized to other settings, e.g., where h_l is discrete or where a penalty other than the squared error is used to push $\hat{h}_l$ towards h_l when training the auto-encoder.

2.2.5 Putting It All Together: Architecture and Training Scheme

The above allows us to propose an architecture and training scheme for deep generative models that are also deep auto-encoders, in Algorithm 1.

Algorithm 1 Proposed Architecture and Training Scheme. Each of the L layers parametrizes an encoder f_l and a decoder g_l. When a target on some value (the output of an encoder f_l or decoder g_l) is specified, it may be used to provide a gradient signal for that encoder or decoder. The top-level auto-encoder's ability to be a good generative model could be improved in various ways, e.g. using the walkback procedure (Bengio *et al.*, 2013b) in which we let it go up and down several times with noise injected and then drive the reconstruction $\hat{h}_{L-1}$ towards either h_{L-1} or a nearby training example's h_{L-1} representation.

Sample training example $h_0 = x \sim Q(X)$
for $l = 1 \ldots L$ **do**
  $h_l = f_l(h_{l-1})$
  h_l is a target for $f_l(corrupt(h_{l-1}))$
end for
$\hat{h}_L = h_L$
for $l = L \ldots 1$ **do**
  if $l < L$, $\hat{h}_l$ is a target for h_l
  $\hat{h}_{l-1} = g_l(\hat{h}_l)$
  h_{l-1} is a target for $g_l(corrupt(\hat{h}_l))$
end for

The corruption helps to make both the encoders and decoders contractive. A reasonable choice for the level of corruption is given by the nearest-neighbor distance between examples in the corresponding representation. In this way, the "empty ball" around each training example is contracted towards that example, but we don't want to contract one training example to its neighbor. The corruption may actually not be necessary for the lower layers encoders because they are regularized by the layers above them, but this corruption is certainly necessary for the top layer auto-encoder, and probably for the lower-level decoders as well.

Notice however how the up-going h_l and down-going $\hat{h}_l$ paths are free of noise. This is to obtain as clean as possible of a target. The mathematical derivation of $\hat{h}_l - h_l$ as an estimator of $\frac{\partial \log P(h_l)}{\partial h_l}$ (up to a constant proportional to the corruption level) relies on the noise-free reconstruction (Alain and Bengio, 2013).

In Algorithm 1, "A is a target of $F(b)$" means that $F(b)$ should receive a gradient pushing it to produce an output closer to A. For example, with squared error, the gradient on $F(b)$ would be $F(b) - A$.

Note that the way in which we propose to make a good denoising auto-encoder out of any stack of auto-encoders starting above layer h_l, i.e., capturing $P(h_l)$, is slightly different from the traditional denoising auto-encoder training procedure. It is motivated by the need to train *all of them* (for all l) at once, and by the objective to make *both the encoders and decoders contractive*, whereas the traditional objective only makes their composition (decode(encode(input))) contractive. The last motivation is that we want the above training to have a chance to work *even without back-prop* across layers, i.e., assuming that a training signal on some output of an up-going encoder f_l or down-going decoder g_l is only used for training that layer-wise encoder or decoder. All this is achieved by making each layer-wise auto-encoder pair (f_l,g_l) a good auto-encoder both ways, but only for the data that matters, i.e., either coming from Q or from P. Thus we want $g_l \circ f_l$ to reconstruct h_{l-1} well and we want $f_l \circ g_l$ to reconstruct $\hat{h}_l$ well. There are two ways in which each layerwise encoder becomes contractive: (1) because of the pull towards modes of $P(h_l)$ in the line "$\hat{h}_l$ is a target for h_l", and (2) because of the corruption noise in the line "h_l is a target for $f_l(corrupt(h_{l-1}))$". Note

how the latter is contractive around samples from $Q(H)$. Similarly the layerwise decoders are made contractive around samples from $P(h)$, i.e., the $\hat{h}$. This may be useful in order to map samples in the neighborhood of $\hat{h}_l$ towards $\hat{h}_{l-1}$. This is good because an imperfect $P(H_l)$ (which does not perfectly imitate $Q(H_l)$) will tend to be less peaky (have more entropy, be flatter) then $P(H_l)$, i.e., it will tend to sample points in the neighborhood of those that are likely under $Q(H_l)$, and we would like g_l to map these "mistakes" back towards the "good values" h_l.

2.2.6 Backprop or No Backprop?

In Algorithm 1, we have a reconstruction target h_l for the upper auto-encoder and "matching" target $\hat{h}_l$ for the encoder $\tilde{f}_l(x)$. In traditional auto-encoder and neural network settings, such targets would be back-propagated as much as possible to provide a signal for all the parameters that influence the mismatch between h_l and $\hat{h}_l$. We conjecture here that it is not necessary to back-prop all the way thanks to the particular structure of the deep auto-encoder and the way it is otherwise trained. We provide a justification for this conjecture below.

The main justification for this conjecture arises out of the fact that we are imposing the KL divergence criterion *at every layer*. First, consider the training signal on the encoder $\tilde{f}_l = f_l \circ \tilde{f}_{l-1}$. The lower-level deep encoder $\tilde{f}_{l-1}$ is already receiving a training signal towards making it transform x into a distribution that matches $P(h_{l-1})$ well, and $P(h_l)$ is related to $P(h_{l-1})$ through the decoder g_l, which maps samples from $P(h_l)$ into samples of $P(h_{l-1})$. Each layer of the encoder is trying to transform its distribution into one that is going to be easier to match by the upper layers, flattening curved manifolds a bit better (keep in mind that a completely flat manifold can be modeled by a single linear auto-encoder, i.e., PCA). Hence we conjecture that it is sufficient to only modify f_l (and not necessarily $\tilde{f}_{l-1}$) towards the target $\hat{h}_l$. This is because $\tilde{f}_{l-1}$ itself is going to get its own target through the target propagation of $\hat{h}_l$ into $\hat{h}_{l-1}$. This is analogous to what happens with back-prop: we use the gradient on the activations of an affine layer l to update the weights of that layer, and it is the back-propagation of the gradient on layer l into the gradient on the layer $l-1$ that takes care of the lower layers. In a sense, back-propagating these targets more than one layer would be redundant.

Second, consider the training signal on the upper auto-encoder. The consistency estimation theorem for denoising auto-encoders presented in Bengio *et al.* (2013b) only requires that the last step of the decoder be trained, so long as it has enough capacity to map its input to its target. Furthermore, if the auto-encoder taking h_{l+1} in input is presumably already doing a good job (both in the sense of minimizing reconstruction error and in the sense that $P(H_{l+1})$ and $Q(H_{l+1})$ are close to each other), then g_{l+1} only needs to learn to prefer the actually sampled "data" h_l to other nearby values, i.e., contracting towards the values it sees as training targets.

It might still be the case that back-propagating all the way further helps Algorithm 1 to converge faster to a good model of the raw data x, but the main conjecture we are making is that by providing a training signal at each layer, the proposed training scheme will be less prone to the training difficulties encountered when training very deep non-linear networks with back-propagated gradients. Where we expect the back-propagation through layers to give more of an advantage is in the supervised scenario of Algorithm 4, below.

What we hope is that target propagation can sideskip the difficulties that arise when using back-propagation for credit assignment, when the dependencies to be captured are highly non-linear. We already know that strong non-linearities arise out of the composition of many layers in a deep net or many steps in a recurrent net and make it difficult to learn by back-propagated gradients because the gradients tend to be either very small (gradient vanishing problem) or very large (gradient explosion problem). This has been well studied in the case of recurrent networks (Hochreiter, 1991; Bengio *et al.*, 1994; Hochreiter and Schmidhuber, 1997), and more recently by Pascanu *et al.* (2013). However, more generally, what to do when the non-linearities are so strong that the derivatives are nearly or actually 0 or infinite? This is what happens with discrete activation functions.

A major advantage of not relying on back-prop is therefore that we can now consider the case where the hidden layer representations are discrete or combine discrete and continuous units at each layer. An advantage of discrete representations is that they are naturally *contractive*, since many input values can potentially be mapped to the same output value due to the discretization step. For the same reason, a discrete auto-encoder should be naturally "error-correcting". Discrete (but

distributed) representations might also be the most natural way to represent input data that is itself discrete, such as language data. On the other hand, some types of data and underlying factors are more naturally represented with real values, so we should probably design systems that can capture both continuous and discrete types.

2.2.7 Sampling From the Model

The training criteria (one for each layer) that are being optimized suggest that we can sample from the model in many ways, all providing a possibly different estimator of the data generating distribution (if the encoder/decoder pairs are powerful enough)[1].

Hence, in principle (due to the fact that we are minimizing $KL(Q(X, H_l)||P(X, H_l))$, Eq. 7, or a bound on the log-likelihood), we can choose any level l and generate a sample of x as follows:

1. Sample h_l from $P(h_l)$.
2. Sample x from $P(x|h_l)$, i.e., $x = \tilde{g}_l(h_l)$.

The first step involves sampling from the deep denoising auto-encoder that sees $h_l \sim Q(H_l)$ as training data (and goes up to h_L to encode h_l). As demonstrated in Bengio *et al.* (2013b), that can be achieved by running a Markov chain where at each step we corrupt the previous MCMC sample, encode it, and decode it. In general one should also add noise proportional to the entropy of $P(h_l|h_L)$. We have assumed that decoders were "almost perfect", i.e., that the entropy of $P(h_l|h_L)$ is zero, in which case no noise would need to be added. In practice, during training, there will be some residual reconstruction error even when $h_l \sim Q(H_l)$. In that case, it might be advantageous to sample from $P(h_l|h_L)$ in the reconstruction step, i.e., add the appropriate noise. Note how the effect of that noise is of smearing the distribution one would get otherwise (convolving it with the noise distribution) so as to make sure to include in the support the examples from $Q(H_l)$.

However, as noted in Alain and Bengio (2013), one potential issue with sampling from denoising auto-encoders is that if the amount of corruption is small, then the chain mixes very slowly, and if it is large, then the reconstruction distribution might not be well approximated by a unimodal reconstruction distribution (which is what we are advocating here, since we assume that the decoder is deterministic). What may save the day, as argued in Bengio *et al.* (2013a) and Bengio *et al.* (2013b), is that mixing tends to be much easier when done at higher levels of representation. In the case of this paper, this is readily done by sampling at the top level, i.e., the generative procedure is summarized in Algorithm 2. In fact, if the top-level auto-encoder is *linear*, then its prior is a Gaussian one (see Section 8.1), and we can sample analytically, not requiring a Markov chain. Similarly, if the top-level auto-encoder is an element-wise auto-encoder (i.e., each dimension is auto-encoder separately), this really corresponds to a factorial distribution, and again we can sample analytically without requiring a Markov chain (see Section 8.2). Another interesting direction of investigation is to replace the reconstruction criterion of the penultimate level h_{L-1} by one in which one only tries to reconstruct to a training set near neighbor. This idea is expanded in Section 2.2.8 below.

2.2.8 Allowing the Top-Level to Mix Well with Nearest-Neighbor Reconstruction

In order to allow the top-level to mix well while allowing the reconstruction distribution to be unimodal and factorial (which is what a deterministic reconstruction really is), we propose to train the top-level denoising auto-encoder or GSN with a criterion that is different from the usual reconstruction criterion. This follows from a proposal jointly developed with Laurent Dinh [2]. We call this new criterion the nearest-neighbor reconstruction criterion.

The motivation is that if we inject a lot of noise in the auto-encoder, it will be impossible for the decoder to perfectly reconstruct the input, and that will force it to have a high entropy (sampling

[1] We have not proven that here, but we conjecture that a consistency theorem can be proven, and its proof would hinge on having enough layers to make sure that $P(H_l)$ approaches $Q(H_l)$, for all layers. There are already proofs that deep and thin generative networks with enough layers can approximate any distribution (Sutskever and Hinton, 2008). With enough layers, we conjecture that one should be able to map any distribution to a factorial one.

[2] Personal communication

Algorithm 2 Generative Procedure Associated with the Training Scheme in Algorithm 1.

First, sample h_{L-1}:
Initialize h_{L-1} randomly (or from a recorded h_{L-1} which was sampled from $Q(H_{L-1})$).
for $k = 1 \dots K$ **do**
  $h_{L-1} = g_L(f_L(corrupt(\hat{h}_{L-1}))$
end for
Second, map h_{L-1} to x:
for $l = L-1 \dots 1$ **do**
  $h_{l-1} = g_l(h_l)$
end for
Return $x = h_0$

from $P(x|h)$ will add a lot of noise). However, in order for each stochastic encode/decode step to correspond to a transition of a Markov chain that estimates the data generating distribution as its stationary distribution, it is enough that the decoder deterministically maps the top-level code towards the nearest "mode" of the data generating distribution, i.e., towards the nearest training example. This can be formalized as follows, denoting A for the stochastic transition operator implemented by a stochastic denoising auto-encoder and $Q(X)$ the data generating distribution (in our case we will apply this to the top level of representation). A sufficient condition for A to generate $Q(X)$ as its stationary distribution is that A mixes while applying A to $Q(X)$ leaves $Q(X)$ unchanged.

Formally this means that the following criterion could be minimized:

$$KL(Q(X)||AQ(X)) = -H(Q(X)) - E_{x\sim Q(X)} \log E_{x'\sim Q(X)} A(x|x') \tag{13}$$

where $AQ(X)$ denotes the application of the linear operator A to the distribution $Q(X)$, and $A(x|x')$ is the probability that A puts on generating state x when starting from state x'.

Intuitively, it means that we want any training example x' to be reconstructable from at least some other example x' on which some probable corruption (small amount of noise) would have been applied (in our case, at the top level of representation). The above criterion involves a double sum over the training data, which seems to make it impractical. However, the inner expectation can be approximated by its largest term, which will given by the nearest neighbor of x in representation space. To see this clearly, first, let us introduce a noise source, which most conveniently would be injected at the top level of the hierarchy, in some latent variable z, i.e., A is decomposed into the following three steps: encode x' into $z' = f(x')$, add noise to z' with $z = z' + \text{noise}$, and decode z with $x = g(z)$. Furthermore, assume the noise as a rapidly decaying probability as a function of $||z||$, like the Gaussian noise, favoring the near neighbors of x as candidates for "explaining" it. Now the above criterion reduces to

$$KL(Q(X)||AQ(X)) \approx E_{x\sim Q(X)} \log \arg\min_x P(\text{noise} = f(x') - f(x))Q(x') \tag{14}$$

which in the case where $Q(X)$ is an empirical distribution is just the noise probability for the nearest neighbor x' of x in the space of representations $f(\cdot)$. The noise distribution could potentially be a function of where we are in space and in which case it would roughly have a standard deviation scaled by the nearest neighbor distance (so that from points far from the data, one would be more likely to make big jumps to reach the data). To minimize this criterion, it would be sufficient to bring $f(x)$ and $f(x')$ closer to each other. We see again the same kind of contractive force that we have observed with the reconstruction-based training. As usual, what prevents $f(x)$ from collapsing into one or a few points is simply that it also has to keep all the information about x that allows one to reconstruct x from $f(x)$.

3 Supervised or Semi-Supervised Learning Target Propagation

In this section, we explore how the ideas presented in the previous section can be extended to provide a layer-local training signal in deep supervised networks. This naturally provides a way to train a deep network in both supervised and semi-supervised modes. Each observed example is now assumed to be either an (x, y) input/target pair or a lone input x.

3.1 Factorial Output Distribution

We will first consider the case where y is a low-dimensional object, such as a category (for classification tasks) or a resonably-sized real-valued vector (for regression tasks) for which $P(y|x)$ can be well approximated by a factorial distribution, given x. In this case we can follow a strategy initiated with Deep Belief Networks (Hinton *et al.*, 2006) and let the top-level auto-encoder (instead of a top-level RBM) model the joint distribution of y and of h_{L-1}, the learned representation of x.

In this context, we consider a prediction of y given x to be simply given by the reconstruction of y given h_L. A denoising auto-encoder (or more generally a Generative Stochastic Network (Bengio *et al.*, 2014)) is naturally trained to reconstruct some of its inputs given the others. The "missing" inputs must be represented in some standard way that allows the auto-encoder to distinguish the "missing" value from likely values. When an input variable is discrete and encoded by a one-hot vector (all zeros except a 1 at the i-th location), a missing value can simply be represented by the vector of all zeros. When an input variable is continuous, missingness can be represented explicitly by a binary vector indicating which input is missing, as in Uria *et al.* (2014). The procedure for predicting y given x is presented in Algorithm 3.

The top-level encoder f_L thus generally takes three arguments, the input representation at the penultimate level, $h_{L-1} = \tilde{f}_{L-1}(x)$, the label y (or 0 if y is missing), and the mask m (a bit indicating whether y is observed, $m = 1$ or missing, $m = 0$). The decoder g_L takes h_L in input and predicts h_{L-1} and y. We denote g_L^y for the part of g_L that predicts y and g_L^h for the part of g_L that predicts h_{L-1}. When y is a category (e.g., represented by a one-hot vector), then as usual with neural networks, one would represent g_L^y with a softmax layer and use cross-entropy (i.e. negative log-likelihood of y given h_L) as a "reconstruction" error.

Algorithm 3 Prediction Procedure Associated with the Supervised Target Propagation Training Scheme in Algorithm 4. It takes x as input and returns a prediction $\hat{y}$.

$h_0 = x$
for $l = 1 \dots L-1$ **do**
  $h_l = f_l(h_{l-1})$
end for
$h_L = f_L(h_{L-1}, 0, 0)$
Return $\hat{y} = g_L^y(h_L)$.

This architecture, combined with the principle already presented in Algorithm 1, gives rise to the training procedure summarized in Algorithm 4 for the supervised or semi-supervised setups.

A way to make sense of what is going on when training the encoders with the label y being given is to consider what distribution is used to provide a target (i.e. a reconstruction) for h_{L-1} in Algorithm 4. For this, we need to generalize a bit the results in Alain and Bengio (2013) regarding the esitmation of $\frac{\partial \log Q(h)}{\partial h}$ via the difference between reconstruction $\hat{h}$ and input h.

Consider an auto-encoder modeling the joint distribution $P(C) = P(A, B)$ of variables A and B, with $C = (A, B)$. For any given fixed $B = b$, the auto-encoder therefore models the conditional distribution $P(A|B)$, as discussed in Bengio *et al.* (2013b). The difference between the reconstruction $\hat{a}$ of a and a itself is thus the models view of $\frac{\partial \log P(A=a|B=b)}{\partial a}$.

In our case, when we "clamp" y to its observed value, what we get in $\hat{h}_{L-1} - h_{L-1}$ is the top auto-encoder's estimated $\frac{\partial \log P(h_{L-1}|y)}{\partial h_{L-1}}$, when h is continuous. In the discrete case, we have argued above (but it remains to demonstrate formally) that $\hat{h}_{L-1}$ is an estimate by the model of a nearby most likely neighbor of h_{L-1}.

Extending the argument to lower layers, we see that each $\hat{h}_l$ is an estimate by the upper layers of a value of the l-th level representation that is near h_l and that is more likely than h_l, *given* y. This is the sense in which this procedure is related to back-propagation and deserves the name of *target propagation*.

Algorithm 4 Target Propagation Training Procedure for Stacked Auto-Encoders in Supervised or Semi-Supervised Setups. Once trained, such a deep network can be used for predictions following Algorithm 3.

Sample training example, either (x, y) (labeled) or x (unlabeled).
$h_0 = x$
$m = 1$ if labeled, 0 otherwise (in which case y can take an arbitrary value).
for $l = 1 \dots L-1$ **do**
  $h_l = f_l(h_{l-1})$
  h_l is a target for $f_l(corrupt(h_{l-1}))$
end for
$h_L = f_L(h_{L-1}, m \times y, m)$
h_L is a target for $f_L(corrupt(h_{L-1}), m \times y, m)$
$\tilde{h}_L = h_L$
$(\hat{y}, \hat{h}_{L-1}) = g_L(\hat{h}_L) = (g^y_L(\hat{h}_L), g^h_L(\hat{h}_L))$
if labeled **then**
  y is a target for $g^y_L(corrupt(\tilde{h}_L)))$
end if
h_{L-1} is a target for $g^h_L(corrupt(\hat{h}_L))$
for $l = L-1 \dots 1$ **do**
  $\hat{h}_l$ is a target for h_l
  $\hat{h}_{l-1} = g_l(\hat{h}_l)$
  h_{l-1} is a target for $g_l(corrupt(\hat{h}_l))$
end for

4 Structured Outputs

If y has a complicated non-factorial conditional distribution, given x, then a simple deterministic function $g^y_L(h_L)$ to predict $E[y|h_L]$ is not going to be enough, and we need to find a more powerful way to capture $P(y|x)$.

In that case, we can have two stacks of auto-encoders, one that mostly models the x-distribution, $P(x)$, and one that mostly models the y-distribution, $P(y)$, but with the top-level codes $h^x_{L-1} = \tilde{f}^x_{L-1}(x)$ and $h^y_{L-1} = \tilde{f}^y_{L-1}(y)$ being such that their joint distribution can easily be modeled by a top-level auto-encoder. Note that because this joint distribution might be complex (and not fully captured by the transformations leading from x to h^x_{L-1} and from y to h^y_{L-1}), the top-level auto-encoder itself will probably need to be a deep one (with its own intermediate layers), rather than a shallow one (which might be ok for straightforward classification problems). In other words, whereas each stack computes useful features for x and y separately, useful features for their joint generally requires combining information from both x and y. When y is an explicit one-hot, this does not seem necessary, but in other cases, it probably is.

Sampling from $P(y|x)$ proceeds as one would expect in a conditional auto-encoder, i.e., compute h^x_{L-1}, consider it fixed (clamped), and run a Markov chain on the top-level auto-encoder, resampling only h^y_{L-1} at each step. Then map h^y_{L-1} to y through the deterministic $\tilde{g}^y_{L-1}$.

5 Multi-Modal Modeling

The supervised and semi-supervised setup described in the previous two sections can easily be generalized to multi-modal modeling. In particular, if there are two general modalities, then the setup of the previous section for learning to represent $P(x, y)$ and sampling $P(y|x)$ can be trivially generalized to obtain a way to sample $P(x|y)$.

If there are N modalities $X^{(1)}, \dots X^{(N)}$, then the architecture can be naturally generalized as follows. Have one stack of auto-encoders for each modality, used to transform each modalities data $X^{(t)}$ into a representation where the marginal $P(X^{(t)})$ can be captured easily through some

$P(H^{(t)})$, with $h^{(t)} = \tilde{f}^{(t)}_{L-1}(x^{(t)})$. Then let $h_{L-1} = (h^{(1)}_{L-1}, \ldots h^{(2)}_{L-1}, h^{(1)}_{L-1})$ and model h_{L-1} with another deep stack of generative auto-encoders.

During training, when a subset of modalities is available, encode the missingness of any modality through a missingness mask $m^{(t)}$ for modality t, and use it to turn off the output of $\tilde{f}^{(t)}_{L-1}(x^{(t)})$ when $x^{(t)}$ is missing (just as we did above when y is missing, in the regular supervised case). Train the top-level deep auto-encoder on the given examples, possibly randomly hiding some subset of the observed modalities so as to provide a target for $h^{(t)}$ for those modalities that were available in the training example but that were hidden at the input of the top-level deep auto-encoder. This is a modality-level random masking similar to the way denoising auto-encoders are trained with random masking of the input vector elements (Vincent *et al.*, 2008).

At test time, any subset of modalities can be sampled given a subset of any of the other modalities, using the same approach as described in the previous section: compute $h^{(t)}_{L-1} = \tilde{f}^{(t)}_{L-1}(x^{(t)})$ for the observed modalities, and start a Markov chain for the top-level auto-encoder, keeping the observed modalities clamped, and initializing the missing modalities using the appropriate missingness masks. After samples of the missing modalities are obtained, project them back into their input spaces through $\hat{x}^{(t)} = \tilde{g}^{(t)}_{L-1}(h^{(t)}_{L-1})$.

6 Target Propagation for Recurrent and Recursive Networks

The ideas introduced in the previous sections can be generalized to handle training of recurrent and recursive networks by target propagation.

In both cases, we compute a state s_i that summarizes a subset $x^{t_i}_{\tau_i} = (x_{\tau_i}, x_{\tau_i+1}, \ldots, x_{t_i})$ of the input data, e.g., the past sequence $(x_1, \ldots x_i)$ for a recurrent net, and a subsequence associated with an internal node of a tree, in a recursive net. We can think of the mapping from $x^{t_i}_{\tau_i}$ to s_i as a deterministic encoding, the output of a recursively defined encoder. In addition, when capturing the P distribution of s_i, there is "side information" that can help to reconstruct it (to provide a target for it that will replace back-propagation), i.e., the rest of the sequence. Hence we can imagine an "upper level" auto-encoder that captures the joint of s_i with the implicit representation of the rest of the sequence. That "auto-encoder", when given both s_i and the representations for the rest of the sequence as inputs, should then be able to provide a reconstruction for s_i that points it towards a more likely configuration, given the rest of the sequence. Therefore, we can use this reconstruction as target for training the encoding mapping into s_i, in addition to the constraint that this encoder should be as much as possible invertible by some decoder for the training data.

Here, unlike in our previous discussions, it is unlikely that the decoder will be able to do a nearly perfect job, even on the training examples, simply because the dimension of s_i is fixed while the dimension of $x^{t_i}_{\tau_i}$ is variable. However, if we assume that the state dimension is large enough, we can also assume that a small reconstruction error will be feasible up to some sequence length. For longer sequences, it seems that we must allow the reconstruction to be stochastic, i.e., to add some noise during decoding.

What is the decoder? The decoder should also be recursive, and its basic building block, in the case of a recurrent net, is the map from the next state s_{t+1} to the tuple combining the current state s_t with the current input x_t. To make the decoding easier, one idea would be to assume that the decoder maps the pair (s_{t+1}, x_{t+1}) to the pair (s_t, x_t), i.e., we think of the recurrent net as a generative one that predicts the future given the past. This decoding is still a difficult job because, during training, both x_t and x_{t+1} are fixed by the data. The advantage of this view is that it also tells us how to generate "backwards" in addition to "forward", i.e., by composition of the decoder and sampling the previous element of the sequence from it. Below we consider a very different route, in which we bite the bullet that information is lost and the decoder must be stochastic.

Both to handle the possibility that x_t has structure (is not a simple one-hot vector) and the possibility that decoding it cannot be done by a deterministic mapping, it would be good to decompose the architecture, as suggested in previous sections, into a deterministic transformation part associated to each x_t, mapping it to r_t, and into a generative part that captures the uncertainty in the joint distribution between some part of the sequence and some other part of the sequence.

More precisely, in the case of a recurrent network, we could have a stack of auto-encoders to model that joint distribution, with the lowest-level encodder taking s_{i-1} and r_i as input, and the higher-level encoder producing s_i. Decoding into s_{i-1}, given s_i and r_i, would then be done in one backward pass through the corresponding layer-wise decoders. On the other hand, predicting r_i given s_{i-1} would involve MCMC sampling from the auto-encoder taking (r_i, s_{i-1}) as input, conditioned on s_{i-1} (i.e., keeping it clamped). In fact, to achieve the best possible sampling, one could generate not just r_i but all the future ones as well (going "up" all the way to the "end" of the sequence and reconstructing all the subsequent r's on the way back). Once an r_i is sampled, it can be deterministically mapped to the corresponding x_i.

In both the case of the recurrent net and of the recursive net, we can think of the architecture as a very deep tree-structured auto-encoder with shared weights, the only difference being that in the case of a recurrent net, the tree is unbalanced and is really a chain with dangling leaves. This deep auto-encoding structure allows to resample any part of the sequence given any part (by doing stochastic encode/decode steps in which only the missing elements are resampled).

The fact that the decoder into the pair (r_i, s_{i-1}) cannot in general be "perfect" means that $P(s_{i-1}|s_i)$ and $P(r_i|s_i)$ must have entropy. It remains to be seen how that uncertainty should be modeled, and wether a simple unimodal distribution (like a Gaussian or a factored Bernoulli or some cross-product of such distributions) would be enough to capture the conditional distribution.

What is interesting is that we have potentially removed back-prop from the picture of training a recurrent or recursive network. It would be very interesting to see if target propagation allows to train recurrent networks to capture longer-term dependencies than backprop-based training.

7 Making the Auto-Encoders Perfect

The initial discussion on the layer-wise KL training criterion and the use of deterministic encoders and decoders assumed that we would be able to learn encoder/decoder pairs that are near inverses of each other for inputs that come respectively from Q (for the encoder) or from P (for the decoder). Is that a reasonable assumption? Note that we do not mean that the auto-encoders necessarily invert *any* x and *any* h. Only that they do it almost perfectly for *almost any sample* from respectively Q or P. If $Q(X)$ lives near a low-dimensional manifold, then the encoder can throw away unnecessary dimensions and thus not be invertible for unlikely input configurations.

As argued above, one motivation for considering the extreme case of perfect auto-encoders rather than assuming some stochastic reconstruction distribution $P(x|h)$ and stochastic encoder $Q(h|x)$ is that if we parametrize the auto-encoders in a non-deterministic way, we may end up with numerical difficulties as they become nearly deterministic in at least some dimensions.

Another, more fundamental motivation for considering perfect auto-encoders is that getting an auto-encoder pair to be almost perfect is not really difficult if the code dimension is sufficient, and that what is difficult instead is to make the encoder transform a complicated distribution ($Q(X)$) into a simple one ($Q(H)$), in the sense of being easier to model (which intuitively means "flatter" or more easily factorisable). The proposal of this paper is to make this transformation gradual, with each layer contributing a little in it.

Perfect reconstruction on the data points can be achieved automatically in various ways (maybe not all desirable). For example, if the encoder is an optimization procedure that looks for

$$\tilde{f}(x) \in \mathrm{argmax}_{h \in S} \log P(x|h) \tag{15}$$

then we get perfect reconstruction so long as S is large enough to have a separate value for each x in the training set. For example, if $x \sim Q(X)$ lives on a d-dimensional manifold, then $S = \mathbf{R}^d$ could be sufficient to get perfect reconstruction.

Another interesting example, that is computationally less demanding, is to make each layer of the encoder easily invertible. For example, consider the usual non-linear transformation in neural networks, with $f_i(h_{i-1}) = \tanh(b_i + W_i h_{i-1})$. The hyperbolic tangent is invertible, and if we make W_i an invertible square $d \times d$ matrix, then we can in principle compute the inverse for cost $O(d^3)$. If we choose minibatches of length greater then d, then inverting the weight matrix is of the same order as computing the matrix multiplication. Even better, we might be able to parametrize W_i so

that it is invertible for a cost $O(d^2)$, which is the same as the matrix-vector product. For example if $W_i = LL'$, the product of a lower-diagonal matrix and its transpose, then the inverse of W_i can be computed by forward and backward-substitution in $O(d^2)$. Another interesting possibility is to decompose $W_i = UDV'$ where U and V are maintained nearly orthogonal and D is diagonal. Maintaining exact orthogonality can be expensive, but maintaining approximate orthogonality is easy (U and V just need to be the encoders of linear auto-encoders with squared reconstruction error).

In general, one would expect that one can *learn* encoder/decoder pairs that are near inverses of each other, and we can make that almost a hard constraint because there are many ways in which this can be done, leaving enough degrees of freedom for other desiderata on the auto-encoder, such as making the distribution simpler as we move up the ladder of representations.

Note that in order for the encoder/decoder pairs to be perfectly matched for the distributions that they see, it is important that the layers have sufficient size. If the dimension of h_i is too small relative to the dimension of h_{i-1}, then the decoder will not be able to do a nearly perfect job on the training data. The dimension of h_i can be reduced with respect to that of h_{i-1} only to the extent that the data really lives in a lower-dimensional manifold. Even then, the reduction should be gradual because the work of compression may be best done in a series of gradual non-linear steps. What we recommend is to actually *keep all the layers of the same size*, but use means other than the layer size to obtain a compression and a contraction. We know that the denoising criterion automatically yields a contraction (Alain and Bengio, 2013), so we do not need to impose an explicit one, although it might be interesting to experiment with alternative ways to encourage contraction, such as the contraction penalty of contractive auto-encoders (Rifai *et al.*, 2011).

If we keep all the latent layers of roughly the same size, then we might want to have either the encoder or the decoder equipped with an intermediate hidden layer that is not considered to be part of the set of latent layers h_i. Instead, such an intermediate hidden layer would simply be considered as part of the computation for the layer-wise encoder or decoder, e.g., the encoder is an affine+rectifier transformation and the decoder is an MLP, or vice-versa. We also have indications that if one fixes the decoder (e.g., to some parametric transformation), then the generally optimal encoder is non-parametric (and can be obtained via an optimization, like in sparse coding). Since an optimization is computationally expensive, we could replace it by a high-capacity MLP. If the encoder or the decoder is an MLP, then we still have to use back-prop internally to train it, but we know that training a shallow neural network by back-prop with gradient-based optimization works well, so this is not a big concern.

Then the question is whether it should be the encoder or the decoder that is equipped with a higher capacity (and internal hidden layer), or both. This question should be determined experimentally, but a practical concern is that we would like the recognition path to be fast (and so would brains), in order to be able to make fast inference and quick decisions. That suggests that the encoder should be a "simple" non-linear transformation (like the usual neural network layer) while the decoder should be an MLP, but this should be resolved experimentally.

8 About the Top-Level Auto-Encoder and Avoiding a Top-Level MCMC to Sample

8.1 Linear Top-Level Auto-Encoder

In the special case where the top-level is linear, training it to minimize denoising reconstruction error estimates a multivariate Gaussian prior for $P(h_{L-1})$. One can readily verify that the optimal denoising reconstruction function minus the input h_{L-1} behaves similarly to the gradient of the log-likelihood, $\Sigma^{-1}(h_{L-1} - \mu)$, which pushes back towards the mean in a stronger way in directions of smaller eigenvalue. However, note how the true gradient blows up if some eigenvalues are 0, unless h_{L-1} happens to be already lying on the allowed manifold. But even in that case, we get a numerically unstable result, dividing a 0 (the projection on the 0-eigenvalue direction) by a 0 (the eigenvalue with value 0). Any slight perturbation of h_{L-1} would throw this off. On the other hand, one would clearly get a stable reconstruction if the system is trained as a linear denoising auto-encoder, because it always sees a numerically bounded reconstruction target, and is trained with stochastic variations of h_{L-1} in the first place.

Viewing the top-level auto-encoder as a Gaussian however has the advantage that one can replace the MCMC sampling scheme of general denoising auto-encoders by the analytic sampling scheme of a Gaussian. The auto-encoder weights can be mapped to the Gaussian covariance (and the biases to the mean) by a simple calculation. However, it is unlikely that every input distribution can be mapped to a Gaussian distribution, simply because we know that there are discrete factors typically involved (e.g., multiple disjoint manifolds). What really makes the Gaussian easy to sample, though, is that it affords a completely factorized representation, where the factors are statistically independent of each other.

8.2 Factorial Top-Level Prior Instead?

Hence if we can map to a factorized top-level distribution (possibly with both discrete and continuous variables), then we can generate (x, h) through completely ancestral sampling, where each step is exact, rather than having to rely on an MCMC for the top level. One interesting question is whether every "reasonable" distribution can be mapped through a generally non-linear but invertible transformation into a completely factorized one (and what "reasonable" then entails).

Note that the top-level auto-encoder does not have a prior that regularizes its code layer, unlike the lower auto-encoder layers. This is equivalent to saying that the top-level prior has a zero gradient, meaning that it has a constant probability, i.e., a completely flat probability distribution, such as the uniform or a large variance Gaussian. Forcing a top-level uniform distribution may be too strong, and it seems that a weaker assumption is that the top-level prior is simply factorial. As discussed above, that makes generative sampling very easy and also makes it more likely that the top-level factors have some intrinsincally interesting meaning that can be revealed through visualizations. The idea would thus be that the top-level prior is not really an auto-encoder, or is a "diagonal" one that reconstructs every unit $h_{L,i}$ separately given itself.

If we choose the top level to be an arbitrary factorial distribution, then instead of doing a reconstruction in order to estimate $\frac{\partial \log P(h_L)}{\partial h_L}$, we can just compute analytically this derivative, for continuous hidden units. What should the equivalent target be for discrete top-level variables? A plausible answer is that the target reconstruction for a discrete unit taking values in some set S should simply be the mode of that discrete distribution. If the unit is binary, it just means that the reconstruction is either equal to the input or to its complement, whichever is most probable. One worrysome aspect of this is that when we do this on every bit of the top level discrete units, we get a target reconstruction that may be very far from the actual output. Maybe we should have a *stochastic* reconstruction, which moves away from the current input values, with a propability (for each direction separately) that is proportional to the ratio of the probability of the mode to the current input?

8.3 Parzen Top-Level Instead

Another interesting possibility is to make the top level of the hierarchy a Parzen distribution, with a regularizer that pushes the variances of each component to be as large as possible.

The "reconstruction" of an example x, seen as h_L, is then basically a linear combination of the nearest neighbors, weighted by their relative component probability. More precisely,

$$
\begin{aligned}
P(h_L) &\propto \sum_i e^{-\frac{1}{2}\frac{||h_L-\mu_i||^2}{\sigma^2}} \\
w_i &= \frac{e^{-\frac{1}{2}\frac{||h_L-\mu_i||^2}{\sigma^2}}}{\sum_j e^{-\frac{1}{2}\frac{||h_L-\mu_j||^2}{\sigma^2}}} \\
\frac{\partial \log P(h_L)}{\partial h_L} &= \sum_i w_i \frac{(\mu_i - h_L)}{\sigma^2}
\end{aligned}
\tag{16}
$$

where μ_i are the Gaussian means, i.e., a set of training examples excluding h_L, and σ is the bandwidth of the Parzen windows.

This should push h_L towards the nearest mode as estimated by the Parzen

8.4 MAP vs MCMC for Missing Modalities and Structured Outputs

An interesting observation is that the noise-free target propagation provides an easy way to perform MAP inference in the case of missing modalities or structured outputs. Indeed, if we take $\hat{h}_{L-1} - h_{L-1}$ as pointing in the direction of the gradient of $\log P(h_{L-1})$ (or towards a more probable configuration, in the discrete case), then local ascent for MAP inference can be achieved as outlined in Algorithm 5, by iteratively encoding and decoding at the level of h_{L-1} with the deep auto-encoder sitting on top of h_{L-1}. This will correspond to a *local ascent* to estimate the MAP, in the space of h_{L-1}.

Algorithm 5 MAP inference over some subset of a representation $h_{L-1}^{(\text{missing})}$ (e.g. associated with a structured output target y, or some missing modalities), given the rest, $h_{L-1}^{(\text{observed})}$ (e.g., associatd with an input x, or some observed modalities).

Compute $h_{L-1}^{(\text{observed})}$ for the observed parts of the data, through their respective encoder functions.
Initialize $h_{L-1}^{(\text{missing})}$ (e.g. to some mean value, preferably set to be 0 by construction).
repeat
  Let $\hat{h}_{L-1}$ be the reconstruction of the top-level auto-encoder (with no noise injected) taking $h_{L-1} = (h_{L-1}^{(\text{observed})}, h_{L-1}^{(\text{missing})})$ as input.
  Update $h_{l-1}^{(\text{missing})} = \hat{h}_{L-1}^{(\text{missing})}$ while keeping the observed parts fixed.
until a maximum number of iterations or convergence of $h_{L-1}^{(\text{missing})}$
Map $\hat{h}_{L-1}$ deterministically back into the data space through the associated decoder functions.
Return the resulting predicted missing values.

In general, we are not interested in finding the global MAP configuration,

$$\text{argmax}_{h_{L-1}} \log P(h_{L-1})$$

but rather a conditional MAP, e.g., if we want to predict the MAP output given some input, or if modalities are observed while others are missing and we want to infer a probable value of the missing ones:

$$\text{argmax}_{h_{L-1}^{(\text{missing})}} \log P(h_{L-1}^{(\text{missing})} | h_{L-1}^{(\text{observed})}). \tag{17}$$

This can be achieved simply by clamping the $h_{L-1}^{(\text{observed})}$ to their observed values and only update the $h_{L-1}^{(\text{missing})}$ in the encode/decode iterations. As discussed above, the target associated with a subset of the elements of h can be interpreted as a gradient, so changing only those while keeping the others fixed amounts to a form of gradient ascent for the missing components given the observed components.

Interestingly, the MCMC version is structurally identical, except that noise is injected in the process. By controlling the amount of noise, we actually interpolate between a MAP-like inference and an MCMC-like posterior sampling inference. Note that in many applications, we care more about MAP inference, since a specific decision has to be taken. However, starting with an MCMC-like inference and gradually reducing the noise would give rise to a form of annealed optimization, more likely to avoid poor local maxima of the conditional probability.

9 Conclusions, Questions, Conjectures and Tests

In this paper we have proposed a radically different way of training deep networks in which credit assignment is mostly performed thanks to auto-encoders that provide and propagate targets through the reconstructions they compute. It could provide a biologically plausible alternative to back-prop, while possibly avoiding some of back-prop's pitfalls when dealing with very deep and non-linear or even discrete non-differentiable computations.

This approach derives primarily from the observation that regularized auto-encoders provide in their reconstruction a value near their input that is also more probable under the implicit probability model that they learn.

This approach has first been derived from a training criterion that tries to match the joint of data and latent representations when they are generated upward (from the data generating distribution and upward through the encoder) or downward (from the learned generative model, and mostly going down through the decoders). This criterion is equivalent to the variational bound on likelihood that has previously been used for graphical models with latent variables.

This paper discusses how this idea can be applied to a wide variety of situations and architectures, from purely unsupervised and generative modeling to supervised, semi-supervised, multi-modal, structured output and sequential modeling.

However, many questions remain unanswered, and the main ones are briefly reminded below.

1. **Can we Prove that a Denoising Auto-Encoder on Discrete Inputs Estimates a Reconstruction Delta that is Analogous to a Gradient?** We already know that reconstruction estimates the log-likelihood gradient of the input for denoising auto-encoders with small noise and continuous inputs. What about larger noise? Is there an analogous notion for discrete inputs? A related question is the following, which would be useful to answer if we want to have a factorial top level. **What would be an appropriate reconstruction target in the case of a factorial discrete distribution?**
2. **Are we Better Off with a Factorial Top-Level or with a Denoising Auto-Encoder Top-Level?**

   Related to the previous question, how should we parametrize the top level? An explicit top-level factorized distribution is advantageous because one can sample analytically from it.
3. **Can we Prove that Algorithm 1 is a Consistent Estimator, with Enough Levels?**

   If we provide enough levels, each with dimension of the same order as the input, and if we train each level according to Algorithm 1, are there conditions which allow to recover the data generating distribution in some form?
4. **Do the Level-Wise Targets Help?**

   In Algorithm 1 or 4, we have targets at each layer and we could back-propagate the associated targets all the way or we could only use them for updating the parameters associated with the corresponding layer. Do the intermediate targets make back-prop more reliable, compared to using *only back-prop*? (in the generative case the comparison point would be the variational auto-encoder, while in the supervised case it would be the ordinary supervised deep net).
5. **Can Back-Prop Between Levels be Avoided Altogether?** Following up on the previous question, can we use *only* the propagated targets and no additional back-prop at all (between layers)? We might still want to use back-prop inside a layer if the layer really is an MLP with an intermediate layer.
6. **Can Every Reasonable Distribution be Mapped to a Factorial One?** Is there a transformation $f(x)$, not necessarily continuous, that maps $x \sim Q(X)$ to $h = f(x)$ such that the distribution of h's is factorial, for any or a very large class of data distributions $Q(X)$?
7. **Is Nearest-Neighbor Reconstruction Yielding Better Models?** Instead of reconstructing the clean input, Section 2.2.8 proposes to reconstruct a near neighbor from the training set, minimizing the discrepancy between the representations of the near neighbors in representation space. Since that was the motivation, does that approach yield better mixing and more accurate generative models?
8. **How to Handle Ambiguous Posteriors?** In this paper we have not addressed the question of ambiguous posteriors, i.e., the the data is really generated from factors whose value cannot be completely recovered from the observation x itself. A natural way to handle such ambiguity would be for the encoder to be stochastic, like in the variational auto-encoder (Kingma and Welling, 2014; Rezende *et al.*, 2014), but more thought is needed to consider what this entails.
9. **Encoder or Decoder as MLP, Neither, or Both?** Each layer of the proposed architecture does not have to be the traditional affine layer (composed with point-wise non-linearity). It could be more powerful, e.g., it could be an MLP with its own hidden units. Should we use such powerful layers at all? in the encoder? in the decoder? in both?

10. **Is Corruption Necessary in the Lower Layers?** When training a deep network such as discussed here, should we limit the injection of corruption to the upper layers only (which capture the stochastic aspect of the distribution), while keeping the lower layers determnistic? Instead of a denoising criterion to achieve contraction in the lower layers, an explicit contractive penalty could be used.

11. **Can a Recurrent Network be Trained to Capture Longer-Term Dependencies by Target-Propagation then by Back-Propagation?**

    Can the idea of propagating targets (obtained as reconstructions) be used to replace back-prop for training recurrent nets to capture long-term dependencies?

References

Alain, G. and Bengio, Y. (2013). What regularized auto-encoders learn from the data generating distribution. In *International Conference on Learning Representations (ICLR'2013)*.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers.

Bengio, Y. (2013). Deep learning of representations: Looking forward. Technical Report arXiv:1305.0445, Universite de Montreal.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE T. Neural Nets*.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *NIPS'2006*.

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013a). Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. ACM.

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013b). Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*.

Bengio, Y., Courville, A., and Vincent, P. (2013c). Unsupervised feature learning and deep learning: A review and new perspectives. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*.

Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *ICML'2014*.

Bornschein, J. and Bengio, Y. (2014). Reweighted wake-sleep. Technical report, arXiv preprint arXiv:1406.2751.

Bottou, L. (2011). From machine learning to machine reasoning. Technical report, arXiv.1102.1808.

Carreira-Perpinan, M. and Wang, W. (2014). Distributed optimization of deeply nested systems. In *AISTATS'2014, JMLR W&CP*, volume 33, pages 10–19.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural computation*, **7**(5), 889–904.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. Technical Report arXiv:1406.2661, arxiv.

Gulcehre, C. and Bengio, Y. (2013). Knowledge matters: Importance of prior information for optimization. In *International Conference on Learning Representations (ICLR'2013)*.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, **40**, 185–234.

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

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

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, T.U. Münich.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, **9**(8), 1735–1780.

Kingma, D. P. and Welling, M. (2014). Efficient gradient-based inference through transformations between bayes nets and neural nets. Technical report, arxiv:1402.0480.

LeCun, Y. (1986). Learning processes in an asymmetric threshold network. In F. Fogelman-Soulié, E. Bienenstock, and G. Weisbuch, editors, *Disordered Systems and Biological Organization*, pages 233–240. Springer-Verlag, Les Houches, France.

LeCun, Y. (1987). *Modèles connexionistes de l'apprentissage*. Ph.D. thesis, Université de Paris VI.

Martens, J. (2010). Deep learning via Hessian-free optimization. In *ICML'2010*, pages 735–742.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, **56**, 71–113.

Ozair, S., Yao, L., and Bengio, Y. (2014). Multimodal transitions for generative stochastic networks. Technical report, U. Montreal, arXiv:1312.5578.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *ICML'2013*.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In *NIPS'06*, pages 1137–1144. MIT Press.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. Technical report, arXiv:1401.4082.

Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML'2011*.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *AISTATS'2009*.

Socher, R., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *ICML'2011*.

Srivastava, N. and Salakhutdinov, R. (2014). Multimodal learning with deep boltzmann machines. *Journal of Machine Learning Research*.

Sutskever, I. (2012). *Training Recurrent Neural Networks*. Ph.D. thesis, Departement of computer science, University of Toronto.

Sutskever, I. and Hinton, G. E. (2008). Deep narrow sigmoid belief networks are universal approximators. *Neural Computation*, **20**(11), 2629–2636.

Swersky, K., Ranzato, M., Buchman, D., Marlin, B., and de Freitas, N. (2011). On autoencoders and score matching for energy based models. In *ICML'2011*. ACM.

Tang, Y. and Salakhutdinov, R. (2013). Learning stochastic feedforward neural networks. In *NIPS'2013*.

Uria, B., Murray, I., and Larochelle, H. (2014). A deep and tractable density estimator. In *Proceedings of the 30th International Conference on Machine Learning (ICML'14)*.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation*, **23**(7).

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