JACOBS
UNIVERSITY

Dzmitry Bahdanau, Herbert Jaeger

# Smart Decisions by Small Adjustments: Iterating Denoising Autoencoders

Technical Report No. 32
May 2014

School of Engineering and Science

# Smart Decisions by Small Adjustments: Iterating Denoising Autoencoders

**Dzmitry Bahdanau, Herbert Jaeger**

*School of Engineering and Science*
*Jacobs University Bremen gGmbH*
*Campus Ring 12*
*28759 Bremen*
*Germany*

*E-Mail:* *d.bahdanau@jacobs-university.de*
*http://minds.jacobs-university.de/node/123*

## Abstract

An iterative neural architecture based on repeated application of the Denoising Autoencoder is introduced. The architecture is placed in the family of other approaches involving networks of simple units and iteration at the exploitation stage. It is shown that repeated feeding of a pattern to a Denoising Autoencoder often yields non-trivial sensible improvements of the pattern. This statement is supported by a classification experiment, in which the data transformed by our architecture is shown to be more linearly separable than the original samples.

# Contents

# 1 Introduction

Deep feed-forward neural networks (DNN) have been attracting active attention of the machine learning research community since the advent of greedy layer-wise pretraining [Hinton et al., 2006]. The early approaches relied on such pretraining with different unsupervised methods such as the Restricted Boltzmann Machine (RBM) [Bengio et al., 2007] and various regularized autoencoders [Vincent et al., 2008, Rifai et al., 2011], followed optionally by a supervised finetuning. The later works showed that they can be trained from careful random initializations as well [Glorot and Bengio, 2010, Krizhevsky et al., 2012]. The usage of GPU's for training these architectures pushed the size limits [Ciresan et al., 2010, Krizhevsky et al., 2012] and advanced regularization techniques such as dropout [Hinton et al., 2012b] and fast dropout [Wang and Manning, 2013] were invented to combat overfitting. Consequently many state-of-the-art results in pattern recognition in different domains belong to deep feed-forward architectures [Krizhevsky et al., 2012, Hinton et al., 2012a].

At the same time researchers considered neural architectures with more complex information flow, involving some sort of feed-back and iterative processing. Classical, multidimensional [Graves et al., 2007] and deep [Pascanu et al., 2013] Recurrent Neural Networks (RNN) involve such a kind of processing, however incorporating inputs into the network state at each particular step is still done in a purely feed-forward fashion. A hierarchical composition of random RNN's with top-down control for denoising and classifying temporal patterns is described in [Jaeger, 2014]. Other examples include the Deep Boltzmann Machine (DBM) [Salakhutdinov and Hinton, 2009], where the intractable posterior over the hidden values is replaced by mean-field approximation optimized iteratively, Generative Stochastic Networks (GSN) [Bengio and Thibodeau-Laufer, 2013] that perform MCMC sampling with both bottom-up and top-down projections alternated with a noise insertion.

In this study we contribute to research of iterative neural architectures by considering a simple example: repeated application of a Denoising Autoencoder (DAE) [Vincent et al., 2008]. An autoencoder is a single hidden layer neural network that is trained to reproduce its input. The DAE is trained to reconstruct the input despite corruption performed intentionally before feeding the input to the network. It was originally invented as a better procedure for layer-wise pretraining of the DNN's, producing more sensible features than the usual autoencoders. Later investigations showed other interesting properties of the DAE's. If one iterates a DAE feeding its output with the addition of noise to its input, the resulting sequence of the outputs forms an MCMC sampler for the underlying data distribution [Bengio et al., 2013]. The walkback modification for the training objective, pushing samples from several first steps to the starting point, helped to improve the quality of the samples. In the case of weak Gaussian noise the optimization

problem that the DAE strives to solve is equivalent to learning the score, i.e. the derivative of the data log-likelihood [Alain and Bengio, 2013]. Thus, the repeated feeding of the output to the input without adding any noise can be perceived as approximate gradient ascent in the direction of greater likelihood and the fixed points of such dynamics might be the likelihood's local maxima.

Driven by the theoretical results described in the last paragraph and a general intuition that DAE "improves" the input we investigate the dynamics of DAE's repeated application on the MNIST data set of handwritten digits [LeCun et al., 1998]. We analyze empirically what the fixed points of such dynamics are and show that often they do look like improved input samples. We hypothesize that these fixed points should be easier to classify and verify this hypothesis with a linear classifier and a simple single hidden layer neural network. We employ the walkback technique to fight unpleasant artifacts emerging during dynamics. The combination of strong corruption and several walkback steps allowed us to obtain fixed points that are more linearly separable than the original samples.

The report is structured as follows. Section 2 introduces autoencoders, Section 3 describes related work. In Section 4 our preliminary investigations that eventually led to us to the exploration of the DAE's dynamics are presented. Section 4 describes in detail the considered model and the dataset. Section 5 presents both qualitative and quantitative results. Finally in the last section we summarize our experience, make conclusions and present ideas for future research.

# 2 Autoencoders

The processing of an autoencoder (also called autoassociative network, Diabolo network) in its most generic form is described by the following equations:

$$h = f(W_1 x + b_1) \tag{1}$$

$$o = g(W_2 h + b_2) \tag{2}$$

where $x \in \mathbb{R}^d, h \in \mathbb{R}^n, o \in \mathbb{R}^d$ stand for the input, the hidden units activations and the output correspondingly; the matrices $W_1$, $W_2$ and the vectors $b_1$, $b_2$ are parameters; $f$ and $g$ are functions picked from domain-specific considerations. The functions $f$ and $g$ are usually nonlinear and are hence often called *nonlinearities*. Popular examples for nonlinearities include the sigmoid $\sigma(x) = 1/(1 - e^{-x})$, hyperbolic tangent, the rectifier $rect(x) = max\{0, x\}$. We will also use the symbol $r$ to refer to the whole reconstruction function $o = r(x)$. The autoencoder is trained to reconstruct its input, that is the training objective is:

$$L_{AE}(r) = \mathbb{E}_x \left[ Err(x, r(x)) \right] \tag{3}$$

where the $Err$ function specifies the punishment for bad reconstruction. Common choices for the $Err$ function include the mean square error (MSE) $||h - o||_2^2$ and

the cross-entropy loss (CE) $\sum_i -x_i \log o_i - (1 - x_i) \log(1 - o_i)$ for the cases when both outputs and inputs are guaranteed to be from the $[0, 1]$ interval. A frequent architectural choice is to tie the weights, that is force $W_2 = W_1^T$. In practice the mean in Equation 3 is replaced by the average on the training set and optimized numerically, typically by the stochastic gradient descent (SGD).

Autoencoders are often associated with dimensionality reduction, that is when the number of hidden units $n$ is considerably less than the size $d$ of the input an autoencoder is forced to learn a compact representation of the data in a lower-dimensional space. A related wide-spread understanding is that of a feature extractor when the activations of hidden units $h$ are later used in a machine learning pipeline as features. One might want to have more features than dimensions in the original data, however setting a desirable $n$ does not help in such a situation because then the autoencoder can learn the identity map. Regularized autoencoders such as the Denoising Autoencoder [Vincent et al., 2008] and the Contractive Autoencoder [Rifai et al., 2011] were invented to allow arbitrary number of features. The general idea in all cases is to put some additional constraint in the learning process so that the autoencoder has to extract meaningful features to accomplish its reconstruction task.

In the case of DAE it is done by randomly corrupting the input during the optimization process. Mathematically it changes the optimization objective to

$$L_{DAE}(r) = \mathbb{E}_{\tilde{x}} \left[ Err(x, r(\tilde{x})) \right] \tag{4}$$

where the random variable $\tilde{x}$ corresponds to $x$ after the noise corruption. In case of images at the input popular choices of corruption include additive Gaussian noise and so called salt-and-pepper noise, when random pixels are blackened or whitened.

The regularized autoencoders as a feature extraction method compete in one league with proper generative models of the data such as the RBM and Sparse Coding [Olshausen et al., 1996] that strive to capture the data distribution at the training phase. A question whether regularized autoencoders implicitly learn the distribution as well is explored in [Alain and Bengio, 2013]. It is first shown that in the case of small additive Gaussian noise with variance $\sigma^2$ and MSE reconstruction error the objective (4) of the DAE training has the following asymptotic expansion:

$$L_{DAE}(r) = \left( \mathbb{E} \left[ ||x - r(x)||_2^2 \right] + \sigma^2 \mathbb{E} \left[ ||\frac{\partial r}{\partial x}||_F^2 \right] \right) + o(\sigma^2) \tag{5}$$

where $r$ denotes the whole reconstruction mapping. Then they prove that minimization of the objective (5) yields the following solution:

$$r^*_{\sigma^2}(x) = x + \sigma^2 \frac{\partial \log p(x)}{\partial x} + o(\sigma^2) \tag{6}$$

where $p(x)$ is the unknown data density. Rewriting it as

$$r^*_{\sigma^2}(x) - x \approx \sigma^2 \frac{\partial \log p(x)}{\partial x} \tag{7}$$

we see that under the assumptions needed for (5) going from some pattern $x$ toward $r^*_{\sigma^2}(x)$ is equivalent to a step in the direction of the data log-likelihood gradient. This result served as a motivation for this study. However it is not directly applicable in our case, since we use noise with the variance comparable to the variance of data.

In [Bengio et al., 2013] a simple MCMC sampler from the underlying data distribution based on a DAE is presented. First a DAE is trained as described above. Then a closed loop is formed by feeding the output to the input. The iteration starts with a sample from the training set which is first corrupted in the way used during the training and then pushed through the DAE, the result is again corrupted and pushed through the DAE and so on. The described algorithm works (see the original work for theoretical foundation), but produces rather spurious samples. To improve the quality of samples the *walkback* modification of the DAE training objective is proposed. It consists of adding the reconstruction errors after the first $k$ steps together to make a new objective (see Figure 1)

$$L_{walkback}(r) = \mathbb{E}_{\tilde{x}} \left[ \frac{\sum_{i=1}^{k} Err(x, r^i(\tilde{x}))}{k} \right] \tag{8}$$

where the upper index $i$ in $r^i$ denotes the step number. Hence the DAE is forced to move to the starting point during the following steps of its dynamics. We note, that the walkback training is essentially training of a recurrent network.

In this work we iterate the DAE almost like it was done in [Bengio et al., 2013], except the corruption at the exploitation stage. Formally it means that the following updates are repeated until convergence:

$$\begin{aligned} h^i &= f(W_1 x^{i-1} + b_1) \\ x^i &= g(W_2 h^i + b_2) \end{aligned} \tag{9}$$

where $x^0 = x$. Figure 1 illustrates the described process.

# 3 Related Work

Methods technically close to ours can be found in [Seung, 1997]. In that work a recurrent network is trained to restore an original handwritten digit pattern after

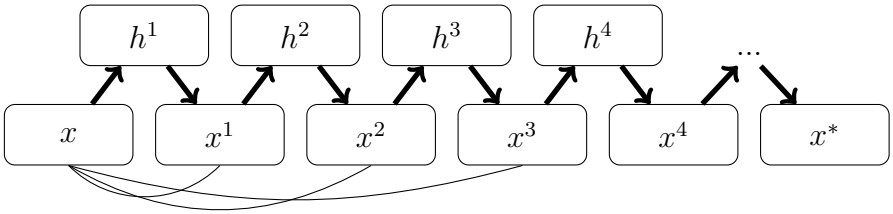


Figure 1: Iterating DAE. $x$ is the original input, $h_i$ are values of hidden layer, $x^i$ are reconstructed inputs at the iteration $i$. $x^*$ symbolizes the fixed point output. The arcs illustrate the structure of the objective for the walkback training procedure for the case $k = 3$ (see Equation 8).

a corruption in form of zeroing a random square. The global structure of the recurrent network is equivalent to what we use, i.e. each step consists of encoding and decoding. The motivation however differs — the author wanted to show that the dynamical system represented by the recurrent network has approximate continuous attractors, that is regions in which the drift towards the real fixed point attractors is negligibly slow. He supported this claim by an observation that "the linearized dynamics has many eigenvalues close to unity". However in our simulations in most of the cases the dynamics converged reasonably fast (usually after several hundred iterations) to a clear fixed point. There are differences in technical details as well: we use fully-connected networks whereas in [Seung, 1997] the neurons had limited square receptive fields, the training methods are different and finally only two iterations of dynamics were considered in that study against hundreds in our work.

Iteration to convergence at the inference stage is used in the DBM [Salakhutdinov and Hinton, 2009]. The DBM is a generalization of the RBM. The RBM is an undirected graphical model consisting of two densely-connected layers of units that are usually called the visible and the hidden layers. As there are no connections between the units of the same layer, the hidden units are conditionally independent given the value of the visible ones. This remarkable property makes inference tractable when the model is learnt. In the DBM multiple layers of hidden units are allowed and exact inference is not tractable any more. Approximate inference is used instead: a mean-field approximation of the posterior is optimized iteratively. For the case of two hidden layers and binary units the following update equations are used:[1]

$$\begin{aligned} h_1^1 &= \sigma(2W_1 x + b_1) \\ h_1^i &= \sigma(W_1 x + W_2^T h_2^{i-1} + b_1) \quad i > 1 \\ h_2^i &= \sigma(W_2 h_1^{i-1} b_2) \quad i > 1 \end{aligned} \tag{10}$$

where $W_1$, $W_2$, $b_1$, $b_2$ are parameters, for details see [Salakhutdinov and Hinton,

[1]The coefficient 2 in the formula for $h_1^1$ is a trick to avoid random initialization of $h_2$, see the cited work for more information.

2009][2]. These equations share some similarity with those used by us (see Equation 9). The respective computation graph has a remarkable structure displayed schematically in Figure 2. Comparing this figure with the Figure 1 one can see two major differences from our approach (excluding the training procedure of course). First, in this work we considered only one hidden layer, the case of two and more layers is in our plans for future research. Second and more important is that the original input is "mixed" into the dynamics at each iteration, whereas our network is left to "think" about the given example on its own.

An attempt to repeat the information flow from Figure 2 in supervised learning traditions is described in the Master thesis [Savard, 2011]. There the phenomenon of the middle layer being recalculated from the upper and the bottom layer was called *relaxation* and the network was allowed to make a fixed a number of relaxation steps (several values up to 8 were tried), with zero relaxation steps corresponding to a classical DNN. The weights in the network were pretrained layer-wise with the DAE criterion, then the supervised finetuning was optionally done. No benefit from the relaxation in terms of classification accuracy on MNIST was observed, except in the case when noise was added at exploitation stage and no finetuning was done. Notable differences between the considered study and our work include all the already mentioned while discussing the DBM. In addition here we iterate the DAE to convergence as opposed to fixed number of steps, the values from the bottom layer instead of the top layer are used for classification. These differences arise from rather different motivations: in this work we focus more on qualitative exploration than on competition with well-established supervised learning methods.

The last study we mention in this section gives an example how recurrent networks can be organized in a hierarchical architecture with top-down feedback [Jaeger, 2014] using filters of neural activity called *conceptors*. A relevant property of conceptors is that it is possible to form their weighted combinations. A toy task is considered in that work: four temporal patterns corrupted with additive Gaussian noise form the possible input space and the problem is to make the right classification decision and output the denoised pattern. The system has to operate online, that is it has to present its current opinion at every step. The proposed solution was a hierarchy of three recurrent neural networks. Each network maintains its current vision of the input signal represented by a conceptor. It transmits the signal filtered with the conceptor it owns to the next layer or to the system output if it is the top-most layer. The top-most layer forms its conceptor from the four corresponding to possible input patterns. The four conceptors are learned before by a supervised procedure. Other layers use weighted combinations of their own conceptors, obtained via *autoconception*, and those formed at the layer above. Thereby the highest layer that actually makes classification controls the lower layers that do the filtering basing on its current classification hypothesis. This

[2]In the cited work the biases $b_1$ and $b_2$ are omitted.

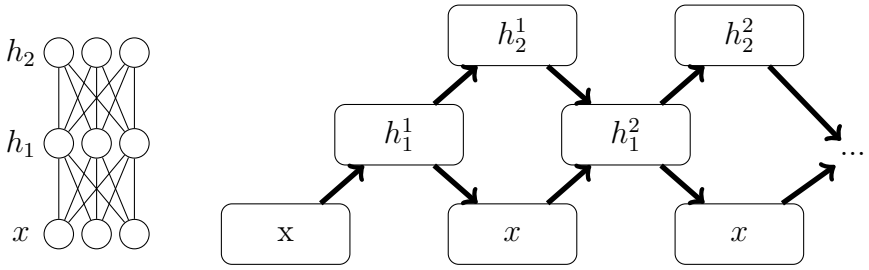


Figure 2: Left: the structure of a DBM with visible units $x$ and with two hidden layers $h_1$ and $h_2$. Right: schematically displayed computations graph of the approximate mean-field inference of the DBM's hidden units. The upper index in $h^i_{1,2}$ stand for the iteration number. The arrows illustrate dependencies, that is for example $h^2_1$ is calculated from $h^1_2$ and $x$.

principle was borrowed in one of our first attempts described in the next section. We refer the reader to the original work for details.

# 4 Preliminary Investigations

In this section we describe the path that led us from the desire to investigate iterative neural architectures to iterating the DAE.

In our first attempt we tried to reconstruct the hierarchical architecture from [Jaeger, 2014] in the context of static patterns. We considered a standard neural network classifier:

$$\begin{aligned} p(y|h) &= softmax(W_2 h + b_2) \\ softmax((v_1, \ldots, v_{|Y|})^T) &= \left(\frac{e^{v_1}}{Z}, \ldots, \frac{e^{v_{|Y|}}}{Z}\right)^T, \; Z = \sum_{i=1}^{|Y|} e^{v_i} \\ h &= \sigma(W_1 x + b_1) \end{aligned} \tag{11}$$

where $Y$ is the set of all labels; $x \in \mathbb{R}^d, h \in \mathbb{R}^n, y \in Y$ are the input vector, the hidden layer activations and the label respectively; the matrices $W_1$ and $W_2$ and the vectors $b_1$ and $b_2$ are parameters; $\sigma$ is the sigmoid nonlinearity. This classifier is in fact a function $f$ that maps the input to the label probabilities, that is $p = f(x)$. Our idea was to repeat the following procedure:

$$\begin{aligned} x^i &= a(x^{i-1}, p^{i-1}) \quad i > 1 \\ p^i &= f(x^i) \end{aligned} \tag{12}$$

where $x^0 = x$ and $p^0 = p$ and $a$ is the input adjusting procedure that takes the current input $x^{i-1}$ and estimated label probabilities $p^{i-1}$. The adjusting procedure $a$ was supposed to "denoise" $x^{i-1}$ using the knowledge from $p^{i-1}$. For instance if only

two classes are plausible for $x^{i-1}$ according to $p^{i-1}$, the procedure $a$ was expected to remove all the traits (if any) pertaining to other classes from $x^{i-1}$ thereby possibly changing the classification decision. In the experiments the procedure was based on applying a weighted combination of conceptors to $x^{i-1}$.

The main problem that we encountered was a total incompatibility of conceptor style filtering and the MNIST data set we used (described in Section 5). The function $a$ was just blurring its input. We found the reason then: it can be shown that a conceptor is a solution for a certain DAE problem (see 5), specifically the one with the Gaussian data and noise distributions. The class-wise Gaussian distribution assumption is entirely false for the handwritten digits. It would be interesting to try the scheme from Equation 12 with proper DAE's.

In our second attempt we tried to organize a gradient descent based iterative procedure at the exploitation stage. We trained an autoencoder on the MNIST training set and considered the reconstruction error $C(x) = ||x - r(x)||^2$ which we called *confusion*. We minimized $C$ as a function of $x$ with gradient descent starting from the classifier input $x$, the result of minimization was then fed to the neural net described by Equation 11. The intuition was that by projecting the input $x$ to the training set on which the misclassification rate had been zero one might circumvent the cause of overfitting: insufficient number of training samples.

Unexpectedly, zero confusion was very easy to achieve with a slight distortion of the image. It turns out that the neural networks are very nonlinear and a small but deliberately chosen shift in the input may strongly change the output [Szegedy et al., 2014]. We suppose that this is because the response of a neural network only makes sense on a very thin manifold it was trained on, and even very close to this manifold one can obtain arbitrary responses.

# 5 Experiment Details

The general scheme of the experiments we conducted was the following: first a DAE autoencoder was trained, then a subset of available samples was repeatedly fed into the DAE as described in Equation (9). The details are explained below.

## 5.1 Data

In this work we use the well-known MNIST data set of handwritten digits [LeCun et al., 1998]. The choice is motivated by its simplicity and wide availability of published results. The data set consists of 70000 samples (60000 in training set and 10000 in test set), each of which is a 28x28 grayscale picture containing a centered handwritten digit. The intensities were divided by 255 to fit in the $[0, 1]$

interval. We used 50000 random samples from the training set for training and the remaining 10000 samples as a validation set.[3]

## 5.2 Architectural Choices

For all the autoencoders we used $f = \sigma$ and $g = \sigma$, i.e. in the hidden and output layers we used the sigmoid nonlinearity. The number of hidden units was 500, which was the largest we could afford using the available hardware. We did not tie weights in most of the experiments, exceptions will be mentioned explicitly. The corruption method was additive Gaussian noise with a varying standard deviation $\sigma$. We used three different levels of corruption: $\sigma = 0.3$ called "low", $\sigma = 0.5$ called "middle" and $\sigma = 0.7$ called "high" (sometimes we will also write "weak", "middle" and "strong" corruption). The optimization objective was MSE (optionally with walkback), so that the conditions necessary to learn the density gradient were formally met, at least with the exception of $\sigma$ being very small.

## 5.3 Training Method

The training method in all cases was the SGD with a minibatch size $b \in \{20, 50\}$, a starting learning rate $\rho = 0.1$ and an annealing rate $\alpha \in \{0.99, 0.999\}$. The minibatches were traversed in a circular order with each full loop called an *epoch*. At the end of each epoch the learning rate was multiplied with $\alpha$. The autoencoders were trained for $N \in \{100, 250, 500\}$ epochs. $k \in \{1, 2, 3, 4, 5\}$ walkback steps were used, with $k = 1$ corresponding to the usual way of training DAE's.

Not all the combinations of hyperparameters listed above were tried, namely the following three series of experiments were conducted:

1. **No walkback.** For each corruption level we tried the following values for the hyperparameters: $b \in \{20, 50\}$, $\alpha \in \{0.99, 0.999\}$, $N \in \{250, 500\}$. All the combinations were considered.
2. **2-3 steps of walkback.** For each noise level we tried $b = 20$, $N \in \{100, 250\}$, $\alpha = 0.9$, as these were the settings that gave the best classification results without walkback.
3. **4-5 steps of walkback.** Only tried for the high corruption level. The hyperparameters were $b = 20$, $N \in \{100, 250\}$, $\alpha \in \{0.99, 0.999\}$.

All the experiments were repeated 5 times with different random seeds. The sources of randomness were the initialization, which was done by the method

[3] In fact we did not need a validation set in our experiments as the reconstruction on both validation and test sets decreased steadily, no early stopping was required.

described in [Glorot and Bengio, 2010] to facilitate good propagation of gradients, and the random corruption.

## 5.4 Iteration

When we had a trained DAE and a set of samples to be iteratively transformed we had to decide on the stopping criterion. To check that the sequence of intermediate outputs $x^i$ converged to a fixed point we tracked the sequence of distances $d_i = ||x_i - x_{i_0}||_2$, where $i_0$ is a fixed iteration number. If $d_i$ stabilized for large enough $i$ at the precision limit of 32-bit floating point operations we made an implication that the sequence $x_i$ converges to $x_{i_0}$.

We used a different stopping criterion to iterate large sets of samples. From considerations of efficiency and simplicity we wanted to process the samples together and stop iteration at the same step for all of them. In preliminary experiments besides $d_i$ we also tracked visually the distance covered at each step of dynamics $l_i = ||x^i - x^{i-1}||$. We observed that $l_i$ was very likely to decrease exponentially after having passed a small enough threshold. That motivated us to use the following approximate criterion: $||x^i - x^{i-1}||_2 < 10^{-4}$. We stopped the iteration after it held for more than 75% percents of the samples; this was done separately for the training, validation and testing sets. The fixed point sets thereby obtained will be later referred to as the *transformed* training, validation and testing sets respectively.

## 5.5 Evaluation

In order to quantitatively measure the quality of the obtained fixed points we trained a logistic regression classifier (see [Murphy, 2012]) on the transformed training set, which was then evaluated on the training and testing sets to give the respective accuracies $acc_{train}$ and $acc_{test}$, i.e. ratios of correct predictions (sometimes also reported in percents). We also tracked the walkback training objective (8) estimated on the training and testing sets, giving the values $MSE^{WB}_{train}$ and $MSE^{WB}_{test}$ (if superscript is omitted it means the usual MSE reconstruction error is reported), and the numbers of explosions, $EX_{train}$ and $EX_{test}$ (see the subsection 6.1 for the explanation what an explosion is).

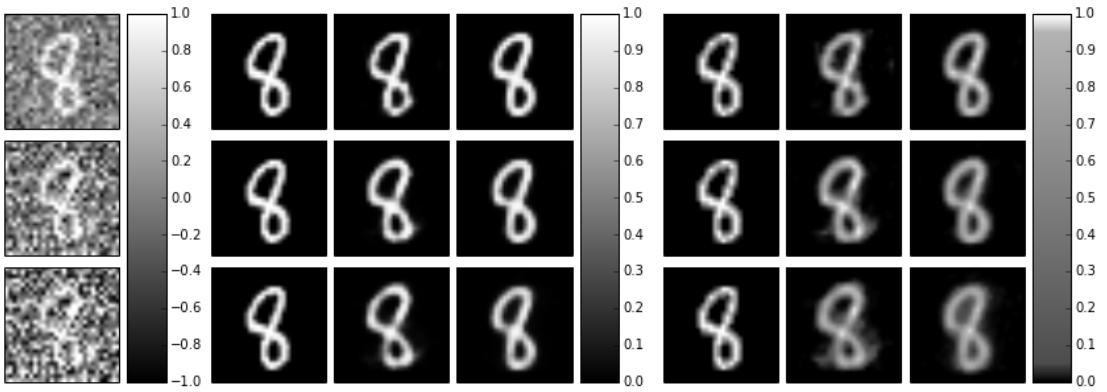

Figure 3: Reconstruction for different corruption levels; the first, second and third rows correspond to the low, middle and high levels. First column: corrupted images, the intensities below -1 and above 1 are suppressed to -1 and 1 respectively. Columns 2-4: original, reconstructed and pushed through the DAE with no noise addition samples. Columns 5-7: same as columns 2-4 but with a special color scheme. The color scheme is highly sensitive for intensities below 0.05 and above 0.95.

# 6 Results

## 6.1 No Walkback

Figure 3 illustrates the functioning of three DAE's trained with different corruption levels. We used a special color scheme to show that after training with stronger corruption a DAE tends to give less contrast responses. For example the original pure black inner pixels of the digit 8 after pushing it through the DAE trained with $\sigma = 0.7$ obtained intensities considerably different from zero, whereas the white pixels of its contour faded. It was not clearly visible in the gray scale.

Figure 4 displays the evolution of the "8" from Figure 3 and 5 other samples. All the four distinct observed outcomes of iteration are shown, namely:

- a pattern with a particular trait loses this trait and becomes a smoothed but very easy to recognize typical representative of its class. For instance a gap in the top of the "0" disappears, same for the curvy tail of the "3". This is the desirable outcome.
- a misclassification happened during dynamics and the pattern leaves its original class and ends as a very clear sample of some other class, e.g. see how the "8" from Figure 3 collapses to one.
- a pattern converges to a something unrecognizable but consisting mostly of black and white pixels as it happened with the "6" for the low corruption level. We will use the nickname "stain" to refer to fixed points like this one.


Figure 4: Intermediate steps in DAE's dynamics. The rows 1 and 4, 2 and 5, 3 and 6 correspond to the low, middle, high corruption levels during training respectively. The columns 1-4 display the image after 0, 1, 25 and 500 iterations, the same for the columns 5-8 and 9-12. The "8" from the rows 1-3 is the one from Figure 3. The other 5 samples were chosen randomly with the constraints that Euclidean distance between the sample and the resulting fixed point must be at least 5 for the three considered models and that they all must represent different classes.

- the final outcome is a gray stain with no white pixels as happened with "4" for the middle corruption level. We call such an outcome "an explosion".

A very remarkable fact, that asks for a nice theoretical explanation, is that for a single DAE *all the fully converged explosions were exactly the same* and had no pixels of high intensity. For instance the explosion from Figure 4 has maximum intensity 0.149. The explosions had made a significant part of outcomes ranging from 2% to 30% before we used the walkback trick. The stains were more frequent for the low corruption level. Another observation is that in general the stronger the corruption during the training was the more gray pixels the fixed points were likely to have. For instance the DAE's used to obtain Figure 4 given the testing set produced fixed points that contained 71.4, 68.4, 65.8 percents of black pixels (intensity less than 0.01) and 10.4, 8.9, 6.3 percent of white pixels (intensity more than 0.95) respectively in the order of increasing corruption level. The reference values of these statistics for the original samples were 80.7 and 8.1 respectively. Finally we note an interesting structure of the fixed points for the high corruption level: a white "skeleton" is surrounded by gray "flesh" much like in an X-ray image.

Figure 5 displays quantitative results: classification accuracies and the frequencies

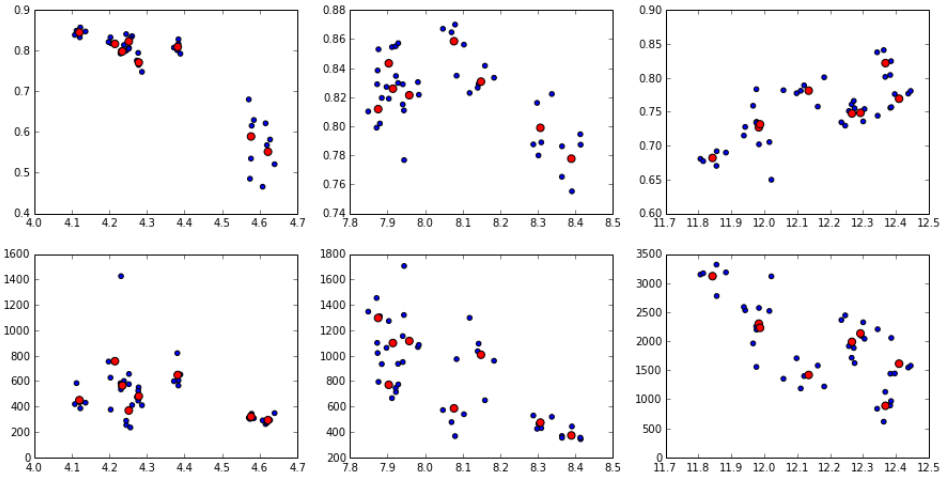

Figure 5: Classification accuracies and explosion frequencies, no walkback. In the first row: x-axis is $MSE_{test}$ , y-axis is $acc_{test}$. In the second row: same for the x-axis, y-axis is the number of explosions in the transformed testing set. The columns from left to right: the low, middle and high corruption levels at the training phase. The blue points are results of single experiments reported for all hyperparameter values, the red points are averages taken over groups of size 5 with similar parameters (see the subsection 5.3).

of explosions for different corruption levels and hyperparameter values. While in the case of the weak corruption better optimization of the training objective yields better accuracy as expected, it is not that clear in the other two cases. Moreover for the strong corruption the opposite is true: the reconstruction error as anticorrelated with the accuracy. For the middle corruption level the picture is more complicated. We also note that the number of explosions grows together with the corruption level. For the middle and high corruption levels it also grows steadily as the reconstruction error decreases reaching very high values of 20-30 percents. It seems that networks trained very well on very noisy examples fail to cope with patterns of the "X-ray" type that were not seen during training causing these to explode.

## 6.2 Walkback

Table 1 displays the quantitative results with the walkback modification. One can see that walkback significantly improved classification accuracies and reduced the frequency of explosions. Its effect was especially strong for the high corruption level. Specifically, when we increased $k$ from 2 to 3 the accuracy gain for the high corruption level was 1.8% versus 1.0% and 0.8% for the middle and low respectively. The further increase of $k$ was done only for strong corruption and was beneficial as well. Our best result was 93.2% accuracy which is a 1% gain

| $k$ | $\sigma$ | $acc_{test}$ | $EX_{test}$ |
|---|---|---|---|
| 1 | 0.3 | 0.823 | 368.8 |
| | 0.5 | 0.859 | 585.6 |
| | 0.7 | 0.822 | 887.8 |
| 2 | 0.3 | 0.858 | 240.4 |
| | 0.5 | 0.899 | 206.2 |
| | 0.7 | 0.893 | 360.4 |
| 3 | 0.3 | 0.866 | 173.2 |
| | 0.5 | 0.909 | 101.4 |
| | 0.7 | 0.915 | 147.6 |

(a) $k \in \{1, 2, 3\}$

| k | N | $\alpha$ | $acc_{train}$ | $acc_{test}$ | $MSE^{WB}_{train}$ | $MSE^{WB}_{test}$ | $EX_{train}$ | $EX_{test}$ |
|---|---|---|---|---|---|---|---|---|
| 4 | 100 | 0.990 | 0.925 | 0.925 | 12.636 | 12.849 | 241.6 | 51.4 |
| | | 0.999 | 0.925 | 0.922 | 13.324 | 13.561 | 221.0 | 51.2 |
| | 250 | 0.990 | 0.931 | 0.930 | 11.948 | 12.173 | 333.4 | 72.2 |
| | | 0.999 | 0.927 | 0.925 | 12.939 | 13.249 | 296.8 | 61.4 |
| 5 | 100 | 0.990 | 0.931 | 0.928 | 12.881 | 13.158 | 95.8 | 26.8 |
| | | 0.999 | 0.927 | 0.924 | 13.823 | 14.158 | 84.4 | 22.0 |
| | 250 | 0.990 | 0.935 | **0.932** | 12.032 | 12.321 | 155.4 | 40.8 |
| | | 0.999 | 0.936 | **0.932** | 13.364 | 13.712 | 102.6 | 25.4 |

(b) $\sigma = 0.7$, $k \in 4, 5$

Table 1: Results with walkback. The notation introduced in Section 5 is used. All the reported values were averaged over five trials.

over usual logistic regression with no regularization.

An interesting observation is that the faster annealing with the rate $\alpha = 0.99$ allowed to optimize the DAE significantly better (up to $\approx 10\%$ difference in $MSE^{WB}_{train}$) with a questionable benefit for classification accuracy and with more explosions in the transformed training and testing sets. This phenomenon was especially pronounced for $k = 5$ and $N = 250$, when the accuracies are similar for both annealing rates and the number of explosions is much lower for $\alpha = 0.99$. On the other hand for both annealing rates in Table 1b we see better correlation of the training objective and classification accuracy as more epochs of training are always helpful. We suppose that different training schedules for the DAE's yielded qualitatively different solutions, but additional investigations are needed.

Finally we note that the performance on the training set is only slightly better than on the testing set, thus overfitting was not an issue.

## 6.3 The Winner

Here we explore one of the two best performing models, specifically the one with $\sigma = 0.999$. Figure 6 shows 12 examples of good and 12 examples of misclassified fixed points. One can see that for relatively easy input samples the DAE actually does some regularization during dynamics, for instance the curved "4" (row 1, column 2) was straightened, for the both "2" (row 1, column 6; row 1, column 12) nice tails were drawn. Another observation is that the most common reason for misclassification of a transformed sample was its de-facto belonging to an other class, see the "3" (row 3, column 1) turning into a very typical eight.

We tried as well to train a neural network classifier on the transformed samples. We only managed to gain additional 0.3% of accuracy, which is not surprising given the evidence from Figure 6.

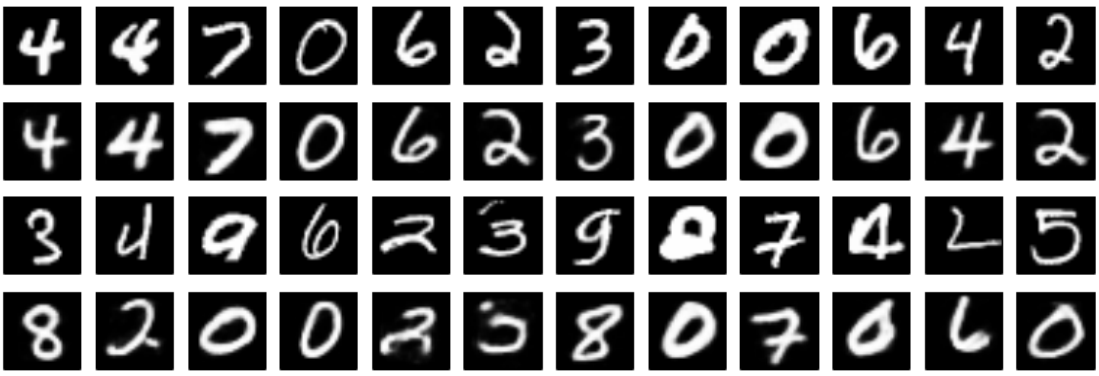

Figure 6: The transformations done by the best DAE. The top row are the original samples chosen randomly with two requirements: a) their transformed versions are correctly classified b) the Euclidean distance is at least 5 between the original and the transformed sample. The second row contains corresponding fixed points. The fourth row are randomly chosen misclassified transformed samples and the third row are the respective originals.

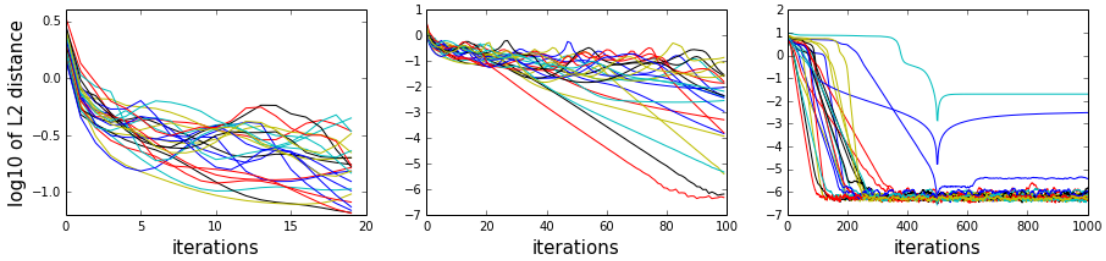


Figure 7: The convergence process of the samples from Figure 6 Left: first 20 step lengths. Middle: first 100 step lengths. Right: Euclidean distance to the 500th steps. Here by step length we mean the Euclidean distance $||x^{i-1} - x^i||_2$ between two consecutive points of a trajectory.

Figure 7 illustrates the dynamics of samples from Figure 6. A typical pattern was that after 50-200 steps a digit got trapped in a basis of attraction of a fixed point attractor and then the distance to the fixed point started decreasing exponentially, which corresponds to straight lines on our log-scale plots. This exponential convergence occurred with different speed but was very stable up to the level guaranteed by 32-bit floating point calculations we relied on.

## 6.4 Other Observations

We tried to tie encoding and decoding weights in DAE's as well, but the results were not better. However inspecting the weight matrices $W_1$ and $W_2^T$ visually we saw that they tend to be very similar. We did not use the walkback trick in the

experiment with tied weights, that might be the key for success that we missed.

# 7 Discussion

In this work using the Denoising Autoencoder model we built a system capable of introducing non-trivial adjustments to simple patterns such as handwritten digits. This was done in a purely unsupervised way with no domain specific insight used. Our system seems to have learnt a lot about handwriting and handwritten digits in particular judging by the way it transforms its input (see Figure 7). Digits transformed by the system were on average easier to deal with for a linear classifier. However the joint performance of the pipeline "DAE iteration + logistic regression" lags a lot behind the $\approx 1\%$ accuracy that current domain-agnostic supervised methods show on the MNIST dataset [Hinton et al., 2012b].

We believe that the key to the intriguing properties of our system is the fact that it is *iterative*, in particular there is *top-down interaction* allowing a transient period of uncertainty (see the middle picture in Figure 7) followed by a decision and rapid convergence to the fixed point. It is not obvious how to train such systems. Here we teach the DAE how to make a single step in the desirable direction. A single step might be a small and rather easy to learn adjustment, but many small steps can constitute complex transformations.

The main question of "teaching how to make a step" approach is where to take lots of examples of good steps from. In this work we started with pairs "corrupted image - original image" and had of a lot of cases of divergence (see Figure 5). We propose the following (not mutually exclusive) explanations for it:

- using samples corrupted with heavy Gaussian noise as input created a bias in the learned models and they could not deal with black-and-white images after such training,
- intermediate samples of the walkback process (that is $x^1, x^2, \ldots, x^{k-1}$) form a large additional training set crucial to learn a good step function,
- the walkback method is a form of regularization of dynamics as it includes propagation of error through time and forces the system to think $k$ steps ahead.

It is an interesting problem for future research to understand the relative weights of these causes. In particular one might restrict the propagation of gradient to only one iteration in depth and eliminate the third explanation, use the distance between the starting sample and sample after $k$ iterations to exclude the second one.

In general usage of strong additive Gaussian noise guarantees that the corrupted

samples will lie outside the manifold of non-zero probability. That being said the main thing we actually taught the network to do is to project a sample to this manifold, not to make small fine improvements as desired. It would be nice to have a corruption process that produces samples corrupted slightly but distinctly and in a sensible way. For handwritten digits examples might include a random stroke addition, partial occlusion, moderate scaling and rotation. In fact such domain-specific corruptions are often used to train a supervised classifier (e.g. in [Ciresan et al., 2010]). The following two possible research directions suggest themselves at this point:

- using domain-specific corruptions one can try to train a smarter adjusting system, such that a relatively simple supervised algorithm requiring few labeled exampled suffices to classify the transformed samples
- one can search for a method to *learn how to corrupt* as well, trying to build a domain-agnostic unsupervised preprocessing scheme

Both these research ideas address one of the most important challenges in machine learning: abundance of unlabeled data and shortage of labeled.

Finally, it would be interesting to repeat our experiment on some other data, with a different architecture, e.g. a deep autoencoder or the DBM inspired scheme from [Savard, 2011], or with the salt-and-pepper corruption method.

# 8 Acknowledgments

For the described experiments the compiler of mathematical expressions Theano [Bergstra et al., 2010] was used.

# References

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

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19.

[Bengio and Thibodeau-Laufer, 2013] Bengio, Y. and Thibodeau-Laufer, É. (2013). Deep generative stochastic networks trainable by backprop. Technical Report arXiv:1306.1091.

[Bengio et al., 2013] Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. *Advances in Neural Information Processing Systems*, 25:899–907.

[Bergstra et al., 2010] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4.

[Ciresan et al., 2010] Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[Graves et al., 2007] Graves, A., Fernandez, S., and Schmidhuber, J. (2007). Multidimensional recurrent neural networks. In *Proceedings of the 2007 International Conference on Artificial Neural Networks*.

[Hinton et al., 2012a] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

[Hinton et al., 2012b] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. Technical Report arXiv:1207.0580.

[Jaeger, 2014] Jaeger, H. (2014). Controlling recurrent neural networks by conceptors. Technical Report 31, Jacobs University Bremen.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT Press.

[Olshausen et al., 1996] Olshausen, B. A. et al. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609.

[Pascanu et al., 2013] Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). How to construct deep recurrent neural networks. Technical Report arXiv:1312.6026.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 833–840.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455.

[Savard, 2011] Savard, F. (2011). Réseaux de neurones à relaxation entraînés par critère d'autoencodeur débruitant.

[Seung, 1997] Seung, H. S. (1997). Learning continuous attractors in recurrent networks. *Advances in Neural Information Processing Systems*, 10:654–660.

[Szegedy et al., 2014] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations*.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, pages 1096–1103. ACM.

[Wang and Manning, 2013] Wang, S. and Manning, C. (2013). Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126.

**Dzmitry Bahdanau** is currently a Master student at Jacobs University Bremen. Before that he studied Computer Science and Applied Mathematics at Belarusian State University, Minsk. His scientific interests lie in the field of machine learning, in particular unsupervised learning and top-down interaction in neural architectures.

**Herbert Jaeger** is Herbert Jaeger is currently an Associate Professor for Computational Science at Jacobs University. His scientific interests lie in the field of machine learning and dynamical systems modeling of neural networks and intelligent agents.