The Potential Energy of an Autoencoder

Hanna Kamyshanska, Roland Memisevic

Abstract—Autoencoders are popular feature learning models, that are conceptually simple, easy to train and allow for efficient inference and training. Recent work has shown how certain autoencoders can be associated with an energy landscape, akin to negative log-probability in a probabilistic model, which measures how well the autoencoder can represent regions in the input space. The energy landscape has been commonly inferred heuristically, by using a training criterion that relates the autoencoder to a probabilistic model such as a Restricted Boltzmann Machine (RBM). In this paper we show how most common autoencoders are naturally associated with an energy function, independent of the training procedure, and that the energy landscape can be inferred analytically by integrating the reconstruction function of the autoencoder. For autoencoders with sigmoid hidden units, the energy function is identical to the free energy of an RBM, which helps shed light onto the relationship between these two types of model. We also show that the autoencoder energy function allows us to explain common regularization procedures, such as contractive training, from the perspective of dynamical systems. As a practical application of the energy function, a generative classifier based on class-specific autoencoders is presented.

Index Terms—Autoencoders, representation learning, unsupervised learning, generative classification

1 Introduction

The difficulty of many classification tasks depends strongly on the representation of the data, more specifically on the choice of *features* used to represent the inputs. Representation learning, or feature learning, tries to simplify classification tasks by learning a good representation automatically from the input data (eg., [1]). Many feature learning methods, like the Restricted Boltzmann Machine (RBM), factor analysis, and many others, are probabilistic models, and training them amounts to maximum likelihood estimation or an approximation thereof.

Since many common models, including the RBM, define probabilities only up to an intractable normalizing constant, approximate maximum likelihood learning typically amounts to optimizing the model only locally, near the training data ([2]). It can be motivated by the observation that many applications rely on the learned *representations* (in the form of hidden unit activations) rather than the probability that the model assigns to the data.

The idea of learning a "landscape" of unnormalized log-probability has been generalized to include arbitrary real-valued functions defined over the data-space, which can be trained to yield small values near the data and large values far away from the data ([3]). These models are commonly referred to as "energy-based models" in the literature ([3]).

Some of the highly successful examples of non-probabilistic feature learning models are autoencoder networks. Autoencoders have been shown to yield state-of-the-art performance in a variety of tasks ranging from object recognition and learning invariant representations to syntactic modeling of text [4], [5], [6], [7], [8], [9], [10]. Learning amounts to minimizing reconstruction error using back-prop.

Autoencoders have been known to be similar in spirit to RBMs in that they learn a good model near the training data only. In fact, if noise is added to the input data during training, minimizing squared error in an autoencoder with sigmoid activations is related to performing *score matching* [7], [8], a common approximation to maximum likelihood learning in the RBM [11]. Score matching itself can be shown to be related to contractive divergence training of the RBM [12]. The relation between RBMs and autoencoders can be extended to other training criteria. For example, [13] suggest training autoencoders using a "contraction penalty" that encourages latent representations to be locally flat, and [14] show that such a regularization penalty applied to real-valued observations allows us to interpret the autoencoder reconstruction function as an estimate of the gradient of the data log probability.

However, beyond the special cases of sigmoid hidden units, or special training criteria that relate autoencoders to RBMs, it has not been clear in general, whether all autoencoders come with an associated energy function, and if so what the form of this energy function is. The restriction of activation function is in particular at odds with the growing interest in unconventional activation functions, like quadratic or rectified linear units, which were shown to work much better than sigmoid units in many tasks.

- *H. Kamyshanska is with Frankfurt Institute for Advanced Studies, Ruth-Moufang-Str. 1, 60438, Frankfurt am Main, Germany. E-mail: kamyshanska@fias.uni-frankfurt.de*
- *R. Memisevic is with University of Montreal, CP 6128, succ Centre-Ville, Montreal H3C 3J7, Canada. E-mail: roland.memisevic@umontreal.ca*

1.1 Autoencoder as dynamical system

In this work, we show how an energy function may be derived for the autoencoder by interpreting it as a *dynamical system*. The view of the autoencoder as a dynamical system was proposed originally by [15], who also demonstrated how de-noising as a learning criterion follows naturally from this perspective.

The dynamical systems perspective allows us to obtain an analytical expression for the energy function of the autoencoder by integration [16]. This makes it possible to assign energies to autoencoders with many different types of activation functions and outputs. We show that the energy function exists irrespective of whether (or how) the model was trained and that its form is not related in any way to any training procedure.

We also show that for autoencoders with sigmoid hidden units and binary or real-valued observations, the energy function is identical to the free energy of the corresponding RBM, showing that autoencoders and RBMs may be viewed as two different ways to derive training criteria for forming the same type of analytically defined energy landscape.

An application of the autoencoder energy surface is generative classification, using autoencoders as the class-specific models. We show how such a classifier yields competitive performance in a variety of tasks in Section 4.2.

2 Autoencoder potential energies

Autoencoders are feed forward neural networks used to learn representations. They map input data to a hidden representation

$$h\big(W\boldsymbol{x}+\boldsymbol{b}_h\big) \tag{1}$$

using an encoder function $h(.)$, from which the data is reconstructed using a linear decoder

$$r(\boldsymbol{x}) = W_{rec}h\big(W\boldsymbol{x}+\boldsymbol{b}_h\big)+\boldsymbol{b}_r \tag{2}$$

For the reconstruction matrix W_{rec} we shall assume that $W_{rec} = W^{\mathrm{T}}$ in the following ("tied weights"). This is common in practice, because it reduces the number of parameters and because related probabilistic models, like the RBM, are based on tied weights, too.

For training, one typically minimizes the average squared reconstruction error for a set of training cases:

$$\frac{1}{N}\sum_i \|r(\boldsymbol{x}^i)-\boldsymbol{x}^i\|_2^2 \tag{3}$$

When the number of hidden units is small, autoencoders learn to perform dimensionality reduction. In practice, it is more common to learn sparse representations by using a large number of hidden units and adding a regularization penalty, such as a "contraction term" [13], to Eq. 3. Alternatively, it is common to reformulate Eq. 3 to reconstruct data from corrupted inputs instead of from the original data during training. The resulting model is known as the de-noising autoencoder [17].

A wide variety of models can be learned, depending on the activation function, number of hidden units and nature of the regularization during training. Autoencoders defined using Eq. 2 with tied weights and logistic sigmoid non-linearity $h(a) = \sigma(a) = \big(1+\exp(-a)\big)^{-1}$ are closely related to RBMs [7], [8], [14].

For binary data, a sigmoid non-linearity is typically used also in the decoder:

$$r(\boldsymbol{x}) = \sigma\big(W_{rec}h\big(W\boldsymbol{x}+\boldsymbol{b}_h\big)+\boldsymbol{b}_r\big) \tag{4}$$

which makes it possible to interpret the outputs as Bernoulli probabilities. For training, the reconstruction error (Eq. 3) is then usually replaced by a cross-entropy loss.

While binary output RBMs make it possible to assign an unnormalized log-probability to the data by integrating over binary hidden units, the analog of an energy function for the autoencoder has been proposed only for real-valued observations. The reason is that the relationship with score matching breaks down in the binary case [14]. As we shall show, the perspective of dynamical systems allows us to attribute the missing link to the lack of symmetry. We also show how we can regain symmetry and thereby obtain a confidence score for binary output autoencoders by applying a log-odds transformation on the outputs of the autoencoder.

2.1 Reconstruction as energy minimization

Since the autoencoder maps an observation $\boldsymbol{x} \in \mathbb{R}^n$ to a reconstruction $r(\boldsymbol{x}) \in \mathbb{R}^n$ it naturally defines a dynamical system [15]. The function $r(\boldsymbol{x})-\boldsymbol{x}$ is a vector field that represents the local displacement that $\boldsymbol{x}$ undergoes as a result of applying the autoencoder reconstruction $r(\boldsymbol{x})$. Repeatedly applying the reconstruction function to an initial point $\boldsymbol{x}$, possibly with a small "inference rate" ϵ, will trace out a non-linear trajectory $\boldsymbol{x}(t)$ in the data-space.

If the number of hidden units is smaller than the number of data dimensions, then the set of fixed points of the dynamical system will be approximately a low-dimensional manifold in the data-space [15]. For overcomplete hiddens it can be a more complex structure. [18], for example, show that for an autoencoder that was trained with a specific denoising or contraction criteria, the reconstruction function will be approximately proportional to the derivative of the log probability of $\boldsymbol{x}$:

$$\boldsymbol{r}(\boldsymbol{x})-\boldsymbol{x} \;\;=\;\; \lambda\frac{\partial \log P(\boldsymbol{x})}{\partial(\boldsymbol{x})}+o(\lambda),\ \lambda\to 0 \tag{5}$$

Running the autoencoder by following the trajectory prescribed by the vector field and starting point $\boldsymbol{x}(0)$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

may also be viewed in analogy to running a Gibbs sampler in an RBM, where the fixed points play the role of a maximum probability "ridge" and where the samples are deterministic not stochastic.

An important observation now is that some vector fields are *gradient fields*, which means they can be written as the derivative of a scalar field [19]. In that case, running the dynamical system can be thought of as performing gradient descent in the scalar field. In analogy to physics, the scalar field may then be thought of as a "potential energy" and the vector field as a corresponding "force".

Evaluating the potential energy for the autoencoder can be useful since it allows us to assess how much the autoencoder "likes" a given input $\boldsymbol{x}$ (up to a normalizing constant which would be the same for any two inputs). The potential energy can therefore play an analogous role to the free energy in an RBM [2], [7]. (In fact, it is identical to the free energy of certain RBMs for sigmoid activation functions as we shall show).

A simple condition for a vector field to be a gradient field is given by the *integrability criterion*, which is a special case of Poincare's Lemma [20]: For some open, simple connected set U, a continuously differentiable function $F : U \to R^n$ defines a gradient field if and only if

$$\frac{\partial F_j(\boldsymbol{x})}{\partial x_i} = \frac{\partial F_i(\boldsymbol{x})}{\partial x_j}, \quad \forall i, j = 1..n \tag{6}$$

In other words, integrability follows from the symmetry of the partial derivatives.

2.2 Computing the energy surface

Consider an autoencoder with shared weight matrix, W, and vectors of hidden/observable biases defined by $\boldsymbol{b}_h$ and $\boldsymbol{b}_r$, respectively, as well a continuously differentiable activation function (e.g. sigmoid, hyperbolic tangent, linear). The integrability criterion holds, since the reconstruction error satisfies

$$\begin{aligned} \frac{\partial (r_m(\boldsymbol{x}) - x_m)}{\partial x_n} &= \sum_j W_{mj} \frac{\partial h(W\boldsymbol{x} + \boldsymbol{b}_h)}{\partial (W\boldsymbol{x} + \boldsymbol{b}_h)} W_{nj} - \delta_{mn} \\ &= \frac{\partial (r_n(\boldsymbol{x}) - x_n)}{\partial x_m} \end{aligned} \tag{7}$$

where δ_{mn} denotes the Kronecker delta.

One way to find the potential energy, whose derivative is $r(\boldsymbol{x}) - \boldsymbol{x}$, is to integrate the vector field (compute its antiderivative).[1] For the autoencoder

$$r(\boldsymbol{x}) = W^{\mathrm{T}} h\big(W\boldsymbol{x} + \boldsymbol{b}_h\big) + \boldsymbol{b}_r \tag{8}$$

1. After computing the energy function, it is easy to check, in hindsight, whether the vector field defined by the autoencoder is in fact the gradient field of that energy function: We just need to compute the derivative of the energy, and check if it is equal to $r(\boldsymbol{x}) - \boldsymbol{x}$. For example, one may check the correctness of Eqs. 12 or 16 by differentiating w.r.t. $\boldsymbol{x}$.

we have

$$\begin{aligned} E(\boldsymbol{x}) &= \int (r(\boldsymbol{x}) - \boldsymbol{x})\, d\boldsymbol{x} \\ &= \int \big(W^{\mathrm{T}} h\big(W\boldsymbol{x} + \boldsymbol{b}_h\big) + \boldsymbol{b}_r - \boldsymbol{x}\big)\, d\boldsymbol{x} \\ &= W^{\mathrm{T}} \int h\big(W\boldsymbol{x} + \boldsymbol{b}_h\big)\, d\boldsymbol{x} + \int (\boldsymbol{b}_r - \boldsymbol{x})\, d\boldsymbol{x} \end{aligned} \tag{9}$$

Define the auxiliary variable $\boldsymbol{u} = W\boldsymbol{x} + \boldsymbol{b}_h$ (which represents the total input of the hidden units). Now using

$$\frac{d\boldsymbol{u}}{d\boldsymbol{x}} = W^{\mathrm{T}} \Leftrightarrow d\boldsymbol{x} = W^{-\mathrm{T}} d\boldsymbol{u} \tag{10}$$

we can write

$$\begin{aligned} E(\boldsymbol{x}) &= \int W^{\mathrm{T}} W^{-\mathrm{T}} h(\boldsymbol{u})\, d\boldsymbol{u} + \boldsymbol{b}_r{}^{\mathrm{T}} \boldsymbol{x} - \frac{1}{2}\|\boldsymbol{x}\|_2^2 + \text{const} \\ &= \int h(\boldsymbol{u})\, d\boldsymbol{u} + \boldsymbol{b}_r{}^{\mathrm{T}} \boldsymbol{x} - \frac{1}{2}\|\boldsymbol{x}\|_2^2 + \text{const} \\ &= \int h(\boldsymbol{u})\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \frac{1}{2}\|\boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \int h(\boldsymbol{u})\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \end{aligned} \tag{11}$$

where the last equation uses the fact that $\boldsymbol{b}_r$ does not depend on $\boldsymbol{x}$.

If $h(\boldsymbol{u})$ is an elementwise activation function, then the final integral is simply the sum over the antiderivatives of the hidden unit activation functions applied to $\boldsymbol{x}$. We can thus compute the energy using the following recipe:

1) Compute the net inputs to the hidden units:
$$\boldsymbol{u} = W\boldsymbol{x} + \boldsymbol{b}_h$$
2) Compute hidden unit activations using the antiderivative $H(\boldsymbol{u})$ of $h(\boldsymbol{u})$ as the activation function.
3) Sum up the resulting activations across all hidden units.
4) Subtract $\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2$. The result is the desired energy (up to integration constant).

2.3 Energy functions for autoencoders with common activation functions

In the following, we derive the energy functions associated with some widely used activation functions. A few example activation functions and their antiderivatives are shown in Figure 1. We only consider autoencoders with real-valued observations in the following. We discuss the modifications necessary to obtain the energy functions for the corresponding models with sigmoid output units in Section 2.5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

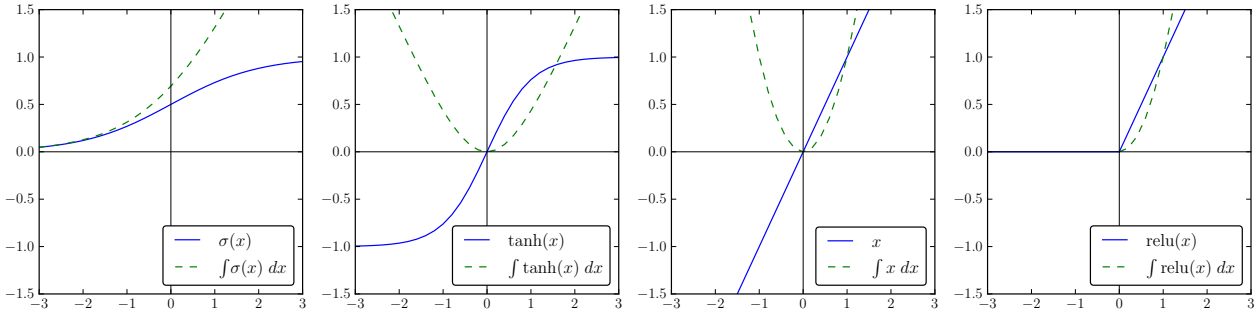

Fig. 1. Some hidden unit activation functions and their integrals.

2.3.1 Sigmoid

The antiderivative of the sigmoid, $h(\boldsymbol{u}) = (1 + \exp(-\boldsymbol{u}))^{-1}$, is given by the "softplus" function $H(\boldsymbol{u}) = \log(1 + \exp(\boldsymbol{u}))$. It follows that the energy function for an autoencoder with sigmoid hiddens takes the form

$$\begin{aligned} E_\sigma(\boldsymbol{x}) &= \int (1 + \exp(-\boldsymbol{u}))^{-1}\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \sum_k \log(1 + \exp(W_{\cdot k}^{\mathrm{T}}\boldsymbol{x} + b_k^h)) - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \end{aligned} \tag{12}$$

Thus, the energy is identical to the free energy in a binary-Gaussian RBM [21]. The analog of the unkown normalizing constant in the RBM is the constant of integration for the autoencoder.

2.3.2 Hyperbolic tangent

Another widely used activation function in neural networks is the hyperbolic tangent:

$$h(\boldsymbol{u}) = \tanh(\boldsymbol{u}) = \frac{\exp(\boldsymbol{u}) - \exp(-\boldsymbol{u})}{\exp(\boldsymbol{u}) + \exp(-\boldsymbol{u})} \tag{13}$$

The corresponding energy function may be written

$$\begin{aligned} E_{\tanh}(\boldsymbol{x}) &= \int_{\boldsymbol{u}} \frac{\exp(\boldsymbol{u}) - \exp(-\boldsymbol{u})}{\exp(\boldsymbol{u}) + \exp(-\boldsymbol{u})}\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \sum_k \log\left(\frac{\exp(u_k) + \exp(-u_k)}{2}\right) - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \sum_k \log\left(\cosh(u_k)\right) - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \qquad (14) \\ &= \sum_k (u_k + \text{softplus}(-2u_k)) - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \qquad (15) \end{aligned}$$

The energy function involves a hyperbolic cosine. Equation 15 is a numerically stable version of 14, whose derivation we show in appendix A.

2.3.3 Linear activation

The antiderivative of the linear function $h(\boldsymbol{u}) = \boldsymbol{u}$, is $\|\boldsymbol{u}\|_2^2/2$. Hence, for the linear autoencoder, whose weight vectors will span the PCA solution [22], we have:

$$\begin{aligned} E_{\text{linear}}(\boldsymbol{x}) &= \int \boldsymbol{u}\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \frac{1}{2}\|\boldsymbol{u}\|_2^2 - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \frac{1}{2}\|W\boldsymbol{x} + \boldsymbol{b}_h\|_2^2 - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \end{aligned} \tag{16}$$

The energy is simply the norm of the latent representation. It is interesting to note that, if we disregard biases and assume $W^{\mathrm{T}}W = I$ (the PCA solution), then $E_{\text{linear}}(\boldsymbol{x})$ turns into the negative squared reconstruction error. Interestingly, this is exactly how one would assign confidence scores to PCA models, such as a PCA based generative classfier.

In Eq. 11 we make use of the invertibility of W^T to derive the energy function in its general form. However, differentiating $E_{\text{linear}}(\boldsymbol{x})$ verifies that for linear hidden units, Eq. 16 will be the correct energy function regardless of shape or invertibility of W^T:

$$\begin{aligned} \frac{\partial E_{\text{linear}}(\boldsymbol{x})}{\partial \boldsymbol{x}} &= W^{\mathrm{T}}W\boldsymbol{x} + W^{\mathrm{T}}\boldsymbol{b}_h + \boldsymbol{b}_r - \boldsymbol{x} \\ &= W^{\mathrm{T}}(W\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_r - \boldsymbol{x} \\ &= r(\boldsymbol{x}) - \boldsymbol{x} \end{aligned} \tag{17}$$

2.3.4 Square activation

The square activation, $h(\boldsymbol{u}) = \boldsymbol{u}^2$, is increasingly common, and it can be shown to be useful for learning video and motion representations [23], [24]. The energy function for the square activation is:

$$\begin{aligned} E_{\text{sq}}(\boldsymbol{x}) &= \int \boldsymbol{u}^2\, d\boldsymbol{u} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \\ &= \sum_k \frac{u_k^3}{3} - \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{b}_r\|_2^2 + \text{const} \end{aligned} \tag{18}$$

2.3.5 Rectified linear (ReLU)

Another increasingly common activation function is the rectified linear activation:

$$\text{relu}(\boldsymbol{u}) = \begin{cases} 0 & \text{if } \boldsymbol{u} < 0, \\ \boldsymbol{u} & \text{else} \end{cases} \tag{19}$$

It can be approximated in the RBM using a stochastic variant called the "noisy rectified linear" activation [25]. Integrating piecewise shows that the antiderivative of $\mathrm{relu}(x)$ is a "half-square" (see also Figure 1): Hence,

$$E_{\mathrm{relu}}(\boldsymbol{x}) = \sum_k (\mathrm{sign}(u_k)+1)\frac{u_k^2}{2} - \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{b}_r\|_2^2 + \mathrm{const}$$

2.3.6 Modulus activation

The piecewise antiderivative of the modulus (absolute value) function, $|\boldsymbol{u}|$, is given by $\mathrm{sign}(\boldsymbol{u})\frac{\boldsymbol{u}^2}{2}$, which yields the energy function

$$E_{\mathrm{abs}}(\boldsymbol{x}) = \sum_k \mathrm{sign}(u_k)\frac{u_k^2}{2} - \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{b}_r\|_2^2 + \mathrm{const} \quad (20)$$

2.3.7 Synchrony autoencoder

The synchrony autoencoder (SAE) was introduced by [26] as an alternative to the square activation for extracting motion features from videos or image pairs. It is based on hidden units whose activation function is a sigmoid applied to squared linear projections of the concatenation of frames in the video. By integrating the sigmoid-of-square activation, one obtains the corresponding energy function

$$E_{\mathrm{SAE}}(\boldsymbol{x}) = \sum_k \log(\exp(u_k^2)+1) - \|\boldsymbol{x}\|_2^2 + \mathrm{const} \quad (21)$$

2.3.8 Maximum activation and Kmeans clustering

By using the maximum activation function and unit norm data and weight vectors, we obtain an autoencoder that is equivalent to Kmeans clustering. Kmeans minimizes the sum of squared distances between observations $\boldsymbol{x}_j$ and cluster centers $\boldsymbol{w}_i$:

$$\sum_{i=1}^{k}\sum_{j:\boldsymbol{x}_j\in C_i}\|\boldsymbol{x}_j-\boldsymbol{w}_i\|_2^2 \quad (22)$$

where C_i is the set of points for which $\boldsymbol{w}_i$ is the closest cluster center ([27]). If cluster centers are normalized, we have

$$\|\boldsymbol{x}_j-\boldsymbol{w}_k\|_2^2 = -2\boldsymbol{x}_j^{\mathrm{T}}\boldsymbol{w}_k + \Omega \quad (23)$$

with $\Omega = \|\boldsymbol{x}_j\|_2^2 + 1$ constant. Define the activation function

$$h_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \langle \boldsymbol{x}, \boldsymbol{w}_i\rangle = \max_j \langle \boldsymbol{x}, \boldsymbol{w}_j\rangle, \\ 0 & \text{else} \end{cases} \quad (24)$$

that chooses the nearest cluster center, as well as a selection function $s(\boldsymbol{x})$ which returns the index of the non-zero hidden unit. The reconstruction in an autoencoder with tied weights can now be written $\boldsymbol{W}^T\boldsymbol{h}(x) = \boldsymbol{w}_{s(\boldsymbol{x})}$ which is the nearest cluster center, whose training criterion is the standard Kmeans objective (Eq. 22). The energy function for this autoencoder is

$$E_{\max}(\boldsymbol{x}) = \int (\boldsymbol{x}-\boldsymbol{w}_{s(\boldsymbol{x})})\, d\boldsymbol{x} = \|\boldsymbol{x}-\boldsymbol{w}_{s(\boldsymbol{x})}\|_2^2 + \mathrm{const}$$

which is simply the squared distance from the nearest cluster center. Note that the energy function is differentiable almost everywhere. As in the case of the linear autoencoder and PCA, the energy E_{max} is the usual confidence score of the Kmeans model.

2.4 Example of a 2-d energy surface

An example of a two-dimensional vector field and the corresponding energy surface is shown in Figure 2. It shows the result of training a contractive autoencoder with 100 hiddens and sigmoid activation on two-dimensional data randomly distributed around a circle. The data, learned vector field and associated energy function are shown in Figures 2**(a)**, **(b)**, **(c)** and **(g)**, respectively. The figure also shows the reconstructions of points drawn from a regular grid (Fig. 2**(d)**) and the corresponding squared reconstruction error (Fig. **(e)** and **(g)**), which illustrates why reconstruction error is not a suitable energy function: It has local minima at *both* the local maxima and the local minima of the energy function and thus it cannot distinguish between "good" and "bad" stationary points of the energy. The unsuitability of squared error as an energy surface is discussed also in [18].

2.5 Binary data

In the case of sigmoid outputs activations, the integrability criterion (Eq. 7) does not hold, because of the lack of symmetry of the derivatives. However, it is possible to regain integrability and obtain an energy function by monotonically transforming the vector space as follows: Use the inverse of the logistic sigmoid (the "log-odds" transformation)

$$\boldsymbol{\xi}(\boldsymbol{x}) = \log\Big(\frac{\boldsymbol{x}}{1-\boldsymbol{x}}\Big) \quad (25)$$

and define the new vector field

$$\begin{aligned} B(\boldsymbol{x}) &= \boldsymbol{\xi}(r(\boldsymbol{x})) - \boldsymbol{\xi}(\boldsymbol{x}) \\ &= \boldsymbol{\xi}\Big(\sigma\big(W^{\mathrm{T}}h\big(W\boldsymbol{x}+\boldsymbol{b}_h\big)+\boldsymbol{b}_r\big)\Big) - \log\Big(\frac{\boldsymbol{x}}{1-\boldsymbol{x}}\Big) \\ &= W^{\mathrm{T}}h\big(W\boldsymbol{x}+\boldsymbol{b}_h\big) + \boldsymbol{b}_r - \log\Big(\frac{\boldsymbol{x}}{1-\boldsymbol{x}}\Big) \end{aligned} \quad (26)$$

The vector field $B(\boldsymbol{x})$ has the same fixed points as $r(\boldsymbol{x})-\boldsymbol{x}$, because invertibility of $\boldsymbol{\xi}(\boldsymbol{x})$ implies

$$r(\boldsymbol{x}) = \boldsymbol{x} \quad \Leftrightarrow \quad \boldsymbol{\xi}(r(\boldsymbol{x})) = \boldsymbol{\xi}(\boldsymbol{x}) \quad (27)$$

Thus running the original and transformed autoencoder will converge to the same representations of $\boldsymbol{x}$.

Although the log-odds transformation is defined (and invertible) only for points $\boldsymbol{x}$ in the *interior* of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

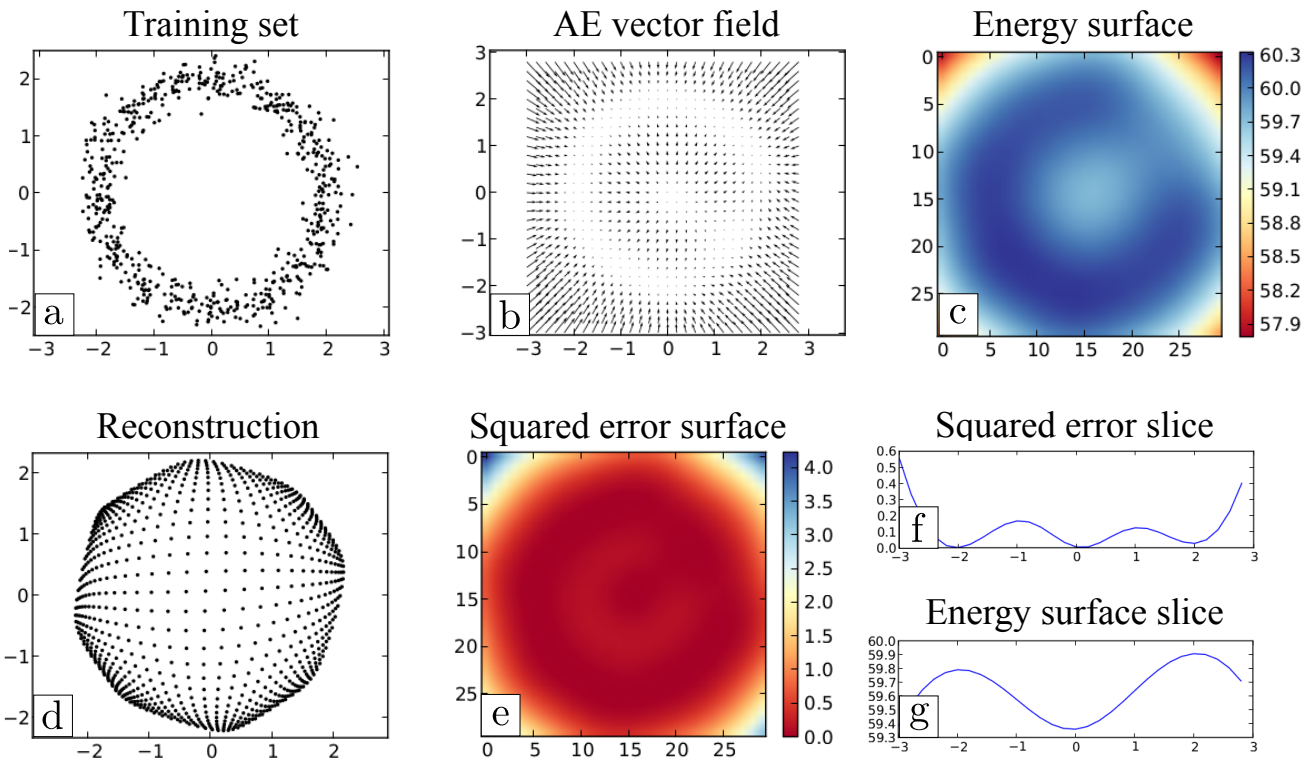

Fig. 2. Example of a learned vector field and the associated energy function. See the text for details.

the hypercube $\boldsymbol{x} \in (0,1)^n$ (in other words, not for binary $\boldsymbol{x}$), any fixed point of the original autoencoder, $r(\boldsymbol{x})$, will reside in the interior, too. The reason is that (even for a binary input vector $\boldsymbol{x}$) the sigmoid outputs of the autoencoder will yield a reconstruction inside the hypercube, as a sigmoid unit can never attain the values 0 or 1. As we show below, the *energy* associated with the transformed autoencoder, is well-defined also for binary data.

By integrating $B(\boldsymbol{x})$, we obtain the energy

$$\begin{aligned} E(\boldsymbol{x}) &= \int h(\boldsymbol{u})\, d\boldsymbol{u} + \boldsymbol{b}_r{}^{\mathrm{T}}\boldsymbol{x} \\ &- \sum_j \left(\log\left(1 - x_j\right) + x_j \log\left(\frac{x_j}{1 - x_j}\right)\right) + \text{const} \end{aligned} \tag{28}$$

As compared to the real-valued autoencoder (c.f., Eq. 11) this energy function contains a bias term that is linear rather than quadratic, as well as the additional entropy term

$$\begin{aligned} &- \sum_j \left(\log\left(1 - x_j\right) + x_j \log\left(\frac{x_j}{1 - x_j}\right)\right) \\ &= - \sum_j \left(x_j \log x_j + (1 - x_j) \log\left(1 - x_j\right)\right) \end{aligned} \tag{29}$$

Using the convention $0 \log(0) = 0$ [28], which is based on the fact that $\lim_{x \to 0} x \log(x) = \lim_{x \to 0} -\frac{x^{-1}}{x^{-2}} = 0$ (l'Hopitals rule), this term vanishes for binary data.

2.5.1 Binary-binary autoencoder

Based on the discussion in the previous section, the energy function for an autoencoder with sigmoid hidden and output units, evaluated at strictly binary data, is given by:

$$E_\sigma(\boldsymbol{x}) = \sum_k \log(1 + \exp(W_{.k}^{\mathrm{T}}\boldsymbol{x} + b_k^h)) + \boldsymbol{b}_r{}^T\boldsymbol{x} + \text{const} \tag{30}$$

The energy is identical to the free energy of a binary-binary RBM [2].

Note that in general, hidden unit activation functions do not need to be sigmoid and enter the energy computation using the recipe described in Section 2.2. Furthermore, on data that is not binary but that takes on values between 0 and 1, the entropy-terms in Eq. 28 are not equal to 0 and need to be included in the energy computation.

3 The Laplacian and Divergence of an Autoencoder

We now discuss how the existence of the energy function makes it possible to apply notions from vector calculus, such as the divergence and Laplacian, to the autoencoder. We then show how these notions lend themselves naturally to defining regularization criteria similar to the contractive regularizer proposed by [13], suggesting to re-interpret this criterion from the perspective of dynamical systems.

The divergence of a vector field is a scalar field that measures the outward flux of the vector field at each point [19]. A negative divergence at a point $\boldsymbol{x}$ signifies that the point is a "sink" of the dynamics, a positive divergence that it is a "source". The divergence of

vector field F can be defined as the sum of the first-order derivatives:

$$\mathrm{div}F(\boldsymbol{x}) = \sum_i \frac{\partial F_i(\boldsymbol{x})}{\partial x_i} \tag{31}$$

In a conservative vector field the divergence is equal to the Laplacian of the energy function, which can be defined as the sum of (non-mixed) second derivatives [19]:

$$\big(\mathrm{div}F(\boldsymbol{x}) = \big)\Delta E(\boldsymbol{x}) = \sum_i \frac{\partial^2 E}{\partial x_i^2} \tag{32}$$

To discuss these notions in the context of the autoencoder we first write the autoencoder dynamics compactly as

$$\frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{x}} = r(\boldsymbol{x}) - \boldsymbol{x} = W^{\mathrm{T}} h(W\boldsymbol{x}) - \boldsymbol{x} \tag{33}$$

Here and in the following we absorb the biases into inputs and hidden units to avoid clutter.

The Hessian of the energy function can now be written:

$$\frac{\partial^2 E(\boldsymbol{x})}{\partial \boldsymbol{x}^2} = W^{\mathrm{T}} \mathrm{diag}\big(h'(\boldsymbol{x})\big) W - I_D \tag{34}$$

where h' denotes the first derivative of the hidden activation function applied element-wise, and D is the dimensionality of the data.

It follows that the Laplacian of the energy function, or equivalently the divergence of the autoencoder vector field, can be written:

$$\begin{aligned}
\Delta E(\boldsymbol{x}) &= \mathrm{tr}\big(W^{\mathrm{T}} \mathrm{diag}(h'(W\boldsymbol{x})) W - I_D\big) && (35)\\
&= \mathrm{tr}\big(W^{\mathrm{T}} \mathrm{diag}(h'(W\boldsymbol{x})) W\big) - D && (36)\\
&= \mathrm{tr}\big(\mathrm{diag}(h'(W\boldsymbol{x})) W W^{\mathrm{T}}\big) - D && (37)\\
&= \sum_k h'(\boldsymbol{w}_k^T \boldsymbol{x}) \|\boldsymbol{w}_k\|_2^2 - D && (38)
\end{aligned}$$

Training an autoencoder by minimizing reconstruction error may be thought of as encouraging training data points to be fixed points of the dynamics, as we discussed. However, this by itself is not a suitable training criterion, especially for overcomplete autoencoders, since they can learn the identity mapping, which would turn all points into fixed points.

From the perspective of reconstruction dynamics, a natural way to regularize the autoencoder is to turn training points into a *sink* of the dynamics (thus limiting the total amount of energy). In this case, training points will not only be encouraged to act as a fixed points but also to be the center of a basin of attraction. Intuitively, traversing points that act as sinks amounts to loosing potential energy, so if the energy surface is bounded, the dynamics have to converge. Another perspective is that the only way to allow for perfect reconstruction of the data by overfitting is to learn a perfectly flat energy surface. But at points that are sinks, the curvature of the energy surface will be negative, making it impossible for the energy surface to be flat everywhere.

Formally, a point $\boldsymbol{x}$ will be a sink of the autoencoder dynamics, if the Laplacian $\Delta E(\boldsymbol{x})$, is negative, or equivalently if

$$\sum_k h'(\boldsymbol{w}_k^T \boldsymbol{x}) \|\boldsymbol{w}_k\|_2^2 < D \tag{39}$$

As we show below, contractive regularization [13] may be viewed as a way to satisfy this criterion. In the following, we shall call the autoencoder *absorbing* at data point $\boldsymbol{x}$ if that data point satisfies Eq. 39. One way (though obviously not the best way) to make all points absorbing would be to restrict weights to be sufficiently small (for example using very strong weight decay). For saturating activation functions the autoencoder can be absorbing locally even for large weights, by having data-points saturate hidden units (in particular, those with large weight vectors).

The contractive regularizer [13], which was originally motivated by encouraging the derivatives of hidden units wrt. the inputs to be small, is defined as the Frobenius norm of the Jacobian:

$$\sum_k \big(h'(\boldsymbol{w}_k^T \boldsymbol{x})\big)^2 \|\boldsymbol{w}_k\|_2^2 \tag{40}$$

Up to the squaring of the derivative in Eq. 40 (which is strictly positive for most common autoencoders, so the square will have only a limited effect and not change signs), Eq. 40 is identical to the left-hand side of Eq. 39.

This shows that contractive regularization as defined in [13] may be viewed as a way to encourage the Laplacian of the autoencoder to be small near the data (although an interesting research direction could be regularizing an autoencoder by enforcing Eq. 39 directly rather than using Eq. 40). A similar point can be made for denoising autoencoders, which themselves may be viewed as a way to approximate a contractive regularizer [18]. Eq. 39 also shows under what conditions contractive regularization will succeed in making the autoencoder absorbing at all training examples: the contraction penality (in the form of the left-hand side of Eq. 39) has to be smaller than D, the dimensionality of the data.

As we discussed in Section 2, an autoencoder is guaranteed to have a well-defined energy function if it has tied weights. It is interesting to note that for an autoencoder whose weights are not tied, contractive regularization will encourage the vector field to be conservative. The reason is that encouraging the first derivative to be small and the second derivative to be negative will tend to bound the energy surface near the training data. Training a contractive or denoising autoencoder may thus also be viewed as a way to optimize the Lyapunov stability of the autoencoder dynamics near the data.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

In the following we show how several existing methods are naturally absorbing according the above definition.

3.1 PCA is absorbing everywhere

The PCA reconstruction is given by

$$\frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{x}} = r(\boldsymbol{x}) - \boldsymbol{x} = W^{\mathrm{T}} W \boldsymbol{x} - \boldsymbol{x} \tag{41}$$

It follows that the PCA Laplacian is given by:

$$\Delta E(\boldsymbol{x}) = \mathrm{tr}\big(W^{\mathrm{T}} W\big) - D = \mathrm{tr}\big(W W^{\mathrm{T}}\big) - D = d - D \tag{42}$$

The Laplacian is constant (independent of the data), and it is negative as long as the dimensionality of the hidden layer is smaller than the data dimensionality. Intuitively, this means that there is a projection (onto a subspace).

3.2 Kmeans clustering is absorbing everywhere

By using the selection function $s(\boldsymbol{x})$ which returns the index of the nearest cluster center (cf., Section 2.3.8) we can write the Kmeans clustering reconstruction function as

$$\frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{x}} = r(\boldsymbol{x}) - \boldsymbol{x} = w_{s(\boldsymbol{x})} - \boldsymbol{x} \tag{43}$$

Note that $r(\boldsymbol{x})$ is differentiable almost everywhere (more specifically, it is differentiable everywhere within the Voronoi cell defined by a cluster center). Since within each cell, $w_s(\boldsymbol{x})$ is constant (and its derivative is zero), the Laplacian takes the form

$$\Delta E(\boldsymbol{x}) = \mathrm{tr}\big(-I_D\big) = -D \tag{44}$$

This shows that Kmeans clustering is globally absorbing, independent of the number and the position of cluster centers.

3.3 CD training as optimizing the Laplacian

Contrastive divergence (CD) training is a popular approximation to maximum likelihood learning for RBMs. It is based on approximating the intractable derivative of the log-normalizing constant of the data-log probability by using local samples from the model [2], [29]. In practice this amounts to applying an update rule which increases unnormalized log-probability at the training examples and decreases it near the data. Nearness is defined in practice by applying a Gaussian perturbation around a one-step reconstruction of the training-example.

A common way to define the one-step reconstruction is the mean-field approach, which amounts to inferring hiddens from the data and subsequently re-inferring the data from the hidden units, which is equivalent to applying the sigmoid autoencoder associated with the RBM. As we showed in Section 2.3, the energy function of this autoencoder is identical to the RBM free energy. Like contractive training of the autoencoder, CD training has the effect of turning the training examples into local optima of the energy function. The close relation between RBMs and autoencoders, and their way of locally optimizing the energy surface, may suggest alternative approaches to training RBMs, such as replacing the sampling-based negative phase with a contractive regularizer.

4 Classification with class-specific autoencoders

One application of the ability to assign an energy, or confidence score, to data-points is to turn a set of class-specific autoencoders into a generative classifier. To this end, one may train one autoencoder per class and combine their confidence scores on a new data sample into a classification decision. One advantage of generative classifiers is that they make it possible to add classes at test-time without having to re-train the parameters of the already trained class-specific models. The most common choice of class-specific model are directed probabilistic models, which are typcially combined into a classification decision using Bayes' rule. But it is possible to use undirected models (and, in fact, autoencoders as we discuss below) as the class-specific models instead, as long as these models are combined properly to obtain a well-calibrated classification decision [2], [30].

Figure 3 shows energies that autoencoders with various different activation functions assign to test cases from classes 6 and 9 of the MNISTsmall digit dataset ([31]), after training on the training cases from each class (one model per class). A similar plot specific to RBMs was presented in [2] which, not surprisingly, looks similar to the plot for the autoencoder with sigmoid activation functions. Here, we used contractive autoencoders for training (see Section 4.2 for more details on the learning). The figure shows that all models yield fairly well separated confidence scores, when the apropriate anti-derivatives are used.

Like for RBMs, scores are relative, not absolute, however, due to the lack of the normalizing constant. This means that we can compare the scores that an autoencoder assigns to multiple data-points, but we cannot compare the scores that multiple different autoencoders assign to a single data-point.

One solution is to aggregate class-specific scores within a discriminative classifier, as shown for Restricted Boltzmann Machines by [30]. The "gated softmax"-model proposed in that work contains a parameter tensor holding all class-specific RBM parameters. Instead of using Bayes rule to turn the class-specific probabilities into a classification decision, that model uses the parameter tensor to define class-probabilities directly. [30] show how, in practice, this amounts to computing the RBM free energy for each class, and combining these with a softmax function.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

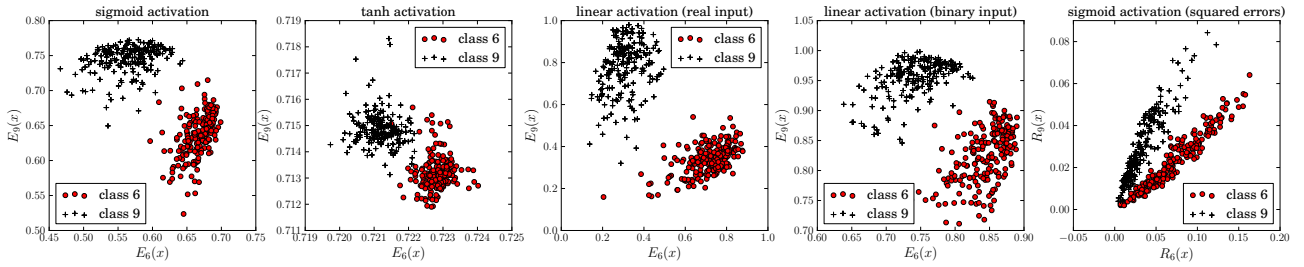

Fig. 3. Examples of class-dependent contractive autoencoder scores distributions and squared reconstruction errors of digits 6 and 9 from the MNISTsmall data set, learned using different activation functions.

Closely related models include the discriminative RBM [32], and the discriminative/generative RBM described in [33].

It is straightforward to apply this approach to autoencoders using their energy function as confidences score. More specifically, denoting the unnormalized energies that the class-specific autoencoders assign to the data as $E_i(\boldsymbol{x}), i = 1, \ldots, K$, we can define the conditional distribution over classes y_i, for example, using multiclass logistic regression as

$$P(y_i|\boldsymbol{x}) = \frac{\exp(E_i(\boldsymbol{x}) + C_i)}{\sum_j \exp(E_j(\boldsymbol{x}) + C_j)} \tag{45}$$

where C_i is the bias term for class y_i. We shall refer to this model as *class-specific autoencoders* (CSAE) in the following.

It is interesting to note that each C_i may be viewed as the normalizing constant of the i^{th} autoencoder, which, since it cannot be determined from the input data, needs to be trained from the labeled data.

Eq. 45 may therefore be viewed as a "*contrastive*" objective function that compares class y_i against all other classes in order to determine the missing normalizing constants of the individual autoencoders. This is reminiscent of noise-contrastive estimation [34], with the difference that the contrastive signal is provided by other classes not by noise. Optimizing the log of Eq. 45 is straightforward using gradient based optimization. Like in the gated softmax model [30], after training the K class specific autoencoders, we may perform discriminative finetuning of the whole model (including all parameters of the K involved autoencoders and the normalizing constants), by optimizing Eq. 45 with respect to these parameters. To obtain their derivatives, one can back-propagate the logistic regression cost.

4.1 Parameter Factorization

We showed in Section 2 that an energy function is guaranteed to exist only for autoencoders with a single hidden layer. We shall now discuss a simple but effective approach to training multilayer models without sacrificing the ability to compute scores, using factorization of a parameter tensor [35], [36].

Note that we may add hidden layers with a *linear activation function*, since we could in principle absorb their weights into the other layers. To define a multilayer autoencoder classification model, we suggest adding linear layers on the bottom and, by symmetry, its transpose on the top. At first sight, it seems that nothing is gained by this proceduce. But since we will be training one autoencoder per class, we may now *tie* the weights of the outer-most layers across classes, to perform class-independent preprocessing of the data. Thus, even though the linear layers do not add any information when training each autoencoder on data from its class, training them with tied weights does.

In many classification tasks such class-specific preprocessing makes sense, because similar features may appear in several classes, so there is no need to learn them separately. Class-specific autoencoders with shared bottom-layer weights can also be interpreted as standard autoencoders whose set of weight matrices is factorized, as proposed by [30] for the gated softmax model based on RBMs.

Fig. 4 shows an illustration of a factored autoencoder. The model parameters are filter matrices $\boldsymbol{W^x}$ and $\boldsymbol{W^h}$ which are shared among all classes, as well as matrices $\boldsymbol{W^f}, \boldsymbol{B}^h$ and $\boldsymbol{B}^r$ which consist of stacked class-dependent feature-weights and bias vectors. Using one-hot encoded labels, $\boldsymbol{t}$, the hidden unit activations and the reconstructions can be written

$$h_k = \sum_f (\sum_i W^{\boldsymbol{x}}_{if} x_i)(\sum_j t_j W^{\boldsymbol{f}}_{jf}) W^{\boldsymbol{h}}_{fk} + \sum_j t_j \boldsymbol{B}^h_{jk} \tag{46}$$

$$r_m = \sum_f (\sum_k W^{\boldsymbol{h}T}_{kf} h_k)(\sum_j t_j W^{\boldsymbol{f}}_{jf}) W^{\boldsymbol{x}T}_{fm} + \sum_j t_j \boldsymbol{B}^r_{jm} \tag{47}$$

Multiplying by $\boldsymbol{t}$, we "mask" the matrices $\boldsymbol{W^f}$, $\boldsymbol{B}^h$ and $\boldsymbol{B}^r$ for each input sample individually, so that only the rows relevant to the current training case remain.

One can interpret this model as a combination of K class-dependent autoencoders with tied parameter sets $\{\boldsymbol{W}^j, \boldsymbol{b}^j_h, \boldsymbol{b}^j_r | j = 1..m\}$, where each weight matrix $\boldsymbol{W}^j$ is factorized into a product of two class-independent (shared) matrices $\boldsymbol{W^x}$ and $\boldsymbol{W^h}$ and one

class-dependent vector $\boldsymbol{w^f}$:

$$W^j_{ik} = \sum_f W^{\boldsymbol{x}}_{if} W^{\boldsymbol{f}}_{jf} W^{\boldsymbol{h}}_{fk} \tag{48}$$

The first encoder-layer ($\boldsymbol{W^x}$) learns the class-independent features, the second layer ($\boldsymbol{W^f_{j\cdot}}$) learns, how dominant these features are in current class and weights them accordingly. Finally, the third layer ($\boldsymbol{W^h}$) learns how to overlay the weighted features to get the hidden representation. All these layers have linear activations except for the last one. Reconstructions and decoder weights take the form

$$r^j_i(\boldsymbol{x}) = \sum_k W^{jT}_{ik} \sigma(h^j_k) + b^j_r \tag{49}$$

$$W^{j\ T}_{ik} = \sum_f W^{\boldsymbol{h}T}_{fk} W^{\boldsymbol{f}}_{jf} W^{\boldsymbol{x}T}_{if} \tag{50}$$

For training in the presence of tied weights, we can update all K autoencoders on data across all classes and use the labels in order to determine for each training case which top-level weights to train. Shared parameters are updated on all training cases. Thus, although the lower layers learn to perform class independent pre-processing, the whole model, including pre-preprocessing weights, will be trained using a single, discriminative objective.

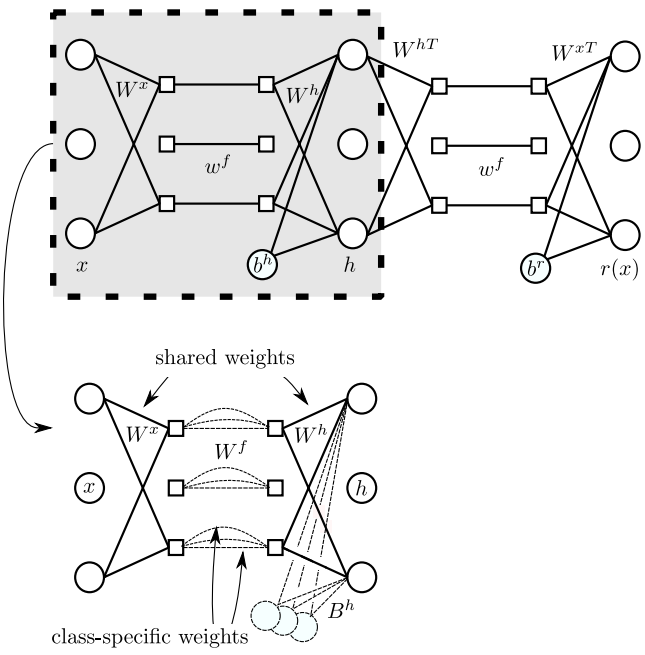

Fig. 4. Factored autoencoder with tied weights. **Top:** a single factored autoencoder; **bottom:** encoder-part of the combination of several autoencoders. Dashed lines represent the class-dependent weights, solid lines class-independent weights.

4.2 Classification experiments

We tested the class-specific autoencoder model (factored and plain) on the regular MNIST dataset [37] and on the deep learning benchmark set [31]. The latter consists of eight datasets, most of which are variations of MNIST. One of the eight datasets, MNISTsmall, is a subset of the ordinary MNIST dataset with only 10000 training samples. All datasets contain gray-scale images of 28×28 pixels with values in the range $[0, 1]$. We used the Python Theano library [38] for most of our experiments. An implementation of the model is available at: http://fias.uni-frankfurt.de/~kamyshanska/aescoring_code.html

In our experiments, we regularized autoencoders with differentiable activation function using contraction penalty [13], and those with non-differentiable activation function using denoising regularization [17]. The factored models were all regularized using a denoising criterion, because the computation of contraction penalties is not feasible for multilayer networks [13].

4.3 Class-specific Pretraining

Although it would be possible to train the log of Eq. 45 starting with a random initialization for all the parameters, [30] report better performance by pretraining all class-specific models separately on data from their own class. We found the same to be true here (see also Section 4.5 for a detailed evaluation).

In our evaluations, we train one set of autoencoders for each number of hiddens (and of factors for the factored model) for pretraining, because at this stage it is not possible to decide which setting will perform the best after discriminative learning. The set of autoencoders per setting (ie., per number of hiddens and of factors) consists of one model per class. We trained each autoencoder separately for the unfactored model, and we trained the autoencoders across all classes simultaneously for the factored model since some parameters are shared between classes, as explained above. We used gradient descent for optimization. In some of our experiments we found that normalizing filters to have equal norm after each gradient step can help stabilize the learning.

For contractive autoencoders the contraction penalty varied between 0.2, 0.5 and 1.0; for denoising autoencoders the corruption level varied between 0.2 and 0.5. Since it is infeasible to perform discriminative finetuning on a very large number of pretrained candidate models, we chose various hyperparameters, such as the number of training epochs, by visual inspection in the pretraining phase. Although a partial search over hyperparameters involving an outer loop for finetuning may slightly improve overall performance, we found the performance to be generally robust to small changes in the pretraining, as long as the pretrained filters looked reasonable. Thus, we do not expect better pretraining to be able to yield a large potential performance improvement. We included the validation sets in the training data

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

used for pretraining to improve the quality of the learned filters (even though this may slightly distort the validation decisions, potentially resulting in suboptimal performance).

4.4 Supervised Finetuning

For discriminative finetuning, we validated over the following hyperparameters: *number of hiddens*, *number of factors (for the factored model)*, *learning rate*, *weight decay*. As weight decay, we used an L_2 penalty on the model parameters. In most cases, we tested 100, 200, 300 and 500 hiddens and factors. For all data sets, in both the pretraining and finetuning phases, we used minibatches of size 100, and momentum of 0.9 for training. We trained all models for 100 epochs using gradient descent (but we found that the best validation cost was often reached during the first 10 epochs).

Finally, after finding the best settings according to the validation data, we finetuned the model on the complete training set (train and validation), and evaluated the resulting model on the test data.

4.5 Classification With and Without Pretraining

We first compared the classification performance of the pretrained and randomly initialized CSAE models on all data sets for linear and sigmoid activations as well as for sigmoid activations with feature sharing. Pretrained models almost always lead to better results on the test data as seen in Figure 6. We also observed that often, the pre-trained filters are already close to the optimum of the discriminative objective, so that just a few epochs of finetuning suffice. Furthermore, finetuning often tends to adjust mainly the scale, rather than the structural details, of the pretrained filters. An illustration of this effect is shown in Figure 5, which shows that filters look similar before and after finetuning. In the example, pretrained filters (left) take on values between -4.72 and 4.67 whereas finetuned filters (right) between -0.64 and 0.63.

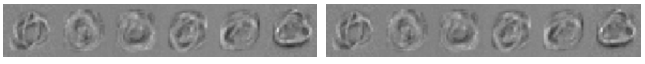

Fig. 5. Initial filters learned by a contractive autoencoder trained on the "0"– digits from MNIST, before supervised finetuning **(left)**, and after supervised finetuning **(right)**.

To get a better understanding of the performance improvement due to finetuning, we compare the test- and the train-errors for autoencoders with linear hidden units in Table 1. It shows that the reason for the lower error rates is data dependent, in that in can be due to the optimization of the training objective (top part of the table), or better generalization (bottom part of the table). The fact that pretraining in some cases

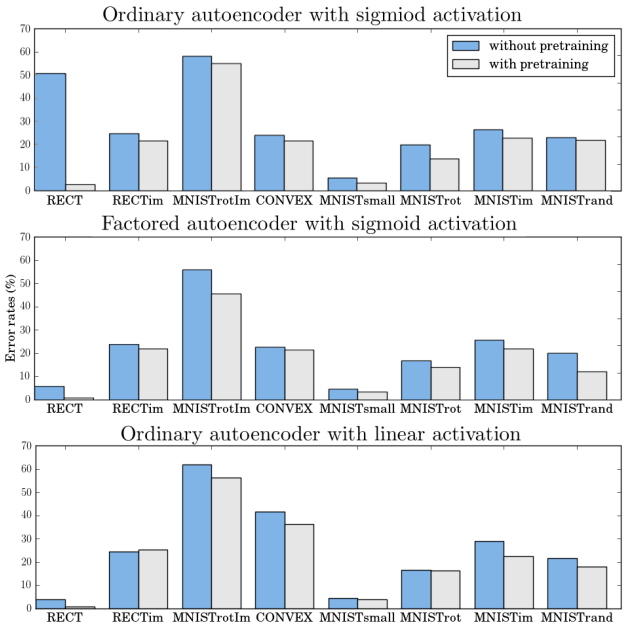

Fig. 6. Test error rates for the CSAE model with and without discriminative pretraining. **Top:** ordinary contractive class-specific AE with sigmoid hidden units. **Middle:** factored denoising AE with sigmoid hidden units. **Bottom:** ordinary contractive class-specific AE with linear hidden units.

simply helps the optimization, may be related to the fact that it may encourage orthonormalization of the weights, as recently discussed in [39].

Overall, our experiments show clearly that unsupervised pretraining generally leads to better results and is preferable over random initialization.

DATA SET	WITH PRETR. TRAIN/TEST	WITHOUT PRETR. TRAIN/TEST
RECTANGLES	**0.08** / **0.84**	0.41 / 3.96
MNISTRAND	**4.16** / **17.96**	20.74 / 21.5
MNISTSMALL	0.04 / **3.91**	**0.02** / 4.29
MNISTROT	4.12 / **16.19**	**0.14** / 16.56
MNISTIMG	0.94 / **22.37**	**0.18** / 29.03
MNISTROTIMG	8.45 / **56.15**	**0.54** / 61.85

TABLE 1
Error rates on train- and test-data sets with and without pretraining, using the linear activation function. Best results shown in bold font.

4.6 Comparison of Activation Functions

We tested the CSAE classification approach on the MNIST data set for a variety of autoencoders: contractive autoencoder with sigmoid and linear activation; denoising autoencoder with modulus activation; factored denoising autoencoder with hyperbolic, modulus, squared, and rectifier activation. Table 2 shows test error rates for these versions of the

		CSAE:	
RBM	3.39	LINEAR	1.99
DRBM	1.81	SQUARED (FACTORED)	2.5
HDRBM	1.28	MODULUS (FACTORED)	2.04
		RELU (FACTORED)	2.35
		MODULUS	1.76
		TANH (FACTORED)	2.02
		SIGM	1.27

TABLE 2
Test-error rates on MNIST for the CSAE Model based on autoencoders with different activation functions.

CSAE model. It also shows error rates for related generative/discriminative models that are based on RBMs [32], including the discriminative RBM (DRBM) and hybrid DRBM (HDRBM). In this experiment, the sigmoid activation yields the lowest error rates; the modulus function also performed well. Somewhat surprising is the comparably good performance of the linear activation, which is the simplest model.

We then evaluated the best performing activation functions on the deep learning benchmark set [31]. The results are shown in Table 3. Again, models with sigmoid activation performed best, with the linear model performing comparably on the RECTANGLES, MNISTsmall and MNISTbackImg tasks. We found the MNISTrand data set in this experiment not to have enough samples per class to learn meaningful class-dependent filters (by inspection). We therefore initialized the final model for this data set using filters from the MNISTsmall data.

DATA SET	LINEAR	MODULUS	SIGM	SIGM (FACT.)
RECTANGLES	**0.84**	4.82	2.72	**0.84**
RECTANGLESIMG	25.3	24.9	**21.45**	22.76
CONVEXSHAPES	36.34	23.18	**21.52**	22.12
MNISTSMALL	3.91	5.17	**3.18**	3.62
MNISTROT	16.19	13.22	**13.11**	14.46
MNISTRAND	17.96	17.08	19.52	**12.64**
MNISTBACKIMG	22.37	23.38	**21.87**	22.77
MNISTROTIMG	56.15	52.88	54.79	**47.14**

TABLE 3
Test-error rates on deep learning benchmark for the basic CSAE model.

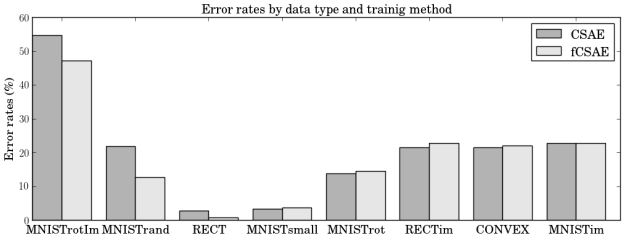

Fig. 8. Test error rates of factored vs unfactored model.

Some learned filters are shown in Figure 7. The figure shows features learned on rotated digits and digits with background images using linear, modulus and sigmoid activation. Interestingly, the filters learned by the different models look qualitatively very different. The modulus autoencoder seems to learn "too much" on MNISTbackImg, while the linear autoencoder captures mostly global structure, such as the shape of the digit "2". On more difficult tasks however, the more local structure seems to be crucial as reflected in the performance table. Figure 7 also shows the shared features from the factored models with sigmoid, modulus and rectifier activations on MNIST. The overall qualitative differences across the models persists.

Another important observation from Table 3 is that factored models are not generally better or worse. It is task-dependent which kind of model to choose. This is also illustrated in Figure 8, which compares the factored versus unfactored variants of the CSAE classifier.

4.7 Comparison of Model Variations

Finally, we trained and compared a variety of variations of the CSAE model:

(hinge): Pretrain the filters, then minimize hinge-loss instead of log-loss (negative log of Eq. 45) for finetuning.

(learn normalization): Train an autoencoder for each class separately, then learn only the normalizing constants by maximizing the conditional log-likelihood (Eq. 45).

(energies): Train an autoencoder for each class separately, then compute for each sample $\boldsymbol{x} \in R^n$ a vector of energies $(E_1(\boldsymbol{x}), \cdots, E_m(\boldsymbol{x}))$, setting the unknown integration constants to zero, and train a linear classifier on labeled energy vectors instead of using the original data [2]. This model performs both normalization and scaling of the pretrained filters.

In our experiments, the log-loss consistently outperformed the hinge loss in all tasks. Methods **energies** and **learn normalization** are very fast due to the small amount of trainable parameters, but they show weaker performance, as shown in Table 4. For comparison, the table also lists the performances of the basic CSAE approach without pre-training.

Two lessons to learn from our experiments are that (i) generative pre-training of each autoencoder on data from its own class is crucial to achieve good performance, (ii) it is not sufficient to adjust merely normalizing and scaling constants, since backpropagating to the filters themselves significantly improves overall performance.

4.8 Comparisons with Other Models

Table 5 shows the classification error rates of CSAE in comparison to various similar models from the literature. We followed the same validation procedure as discussed above. In these experiments, to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2014.2362140, IEEE Transactions on Pattern Analysis and Machine Intelligence

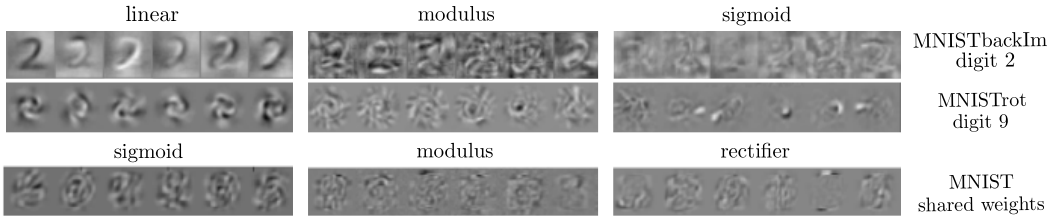

Fig. 7. Filters learned using autoencoders with different activation functions.

DATA SET	ENERGIES	NORMALIZE ONLY	NO PRETR.	CSAE
MNISTSMALL	3.39	3.66	5.53	**3.38**
MNISTROTATION	14.32	20.85	19.95	**13.77**
CONVEX	44.36	33.9	24.02	**21.52**

TABLE 4
Classification results for variations of the model.

DATA	SVM RBF	RBM	DEEP SAA_3	GSM	CSAE
RECTANGLES	2.15	4.71	2.14	**0.56**	0.84
RECTANGLESIM	24.04	23.69	24.05	22.51	**21.45**
CONVEXSHAPES	19.13	19.92	18.41	**17.08**	21.52
MNISTSMALL	**3.03**	3.94	3.46	3.70	**2.61**
MNISTROT	11.11	14.69	**10.30**	11.75	11.25
MNISTIMG	22.61	**16.15**	23.00	22.07	20.15
MNISTRAND	14.58	**9.80**	11.28	10.48	12.64
MNISTROTIMG	55.18	52.21	51.93	55.16	**47.14**

TABLE 5
Test-error rates on the deep learning benchmark data set. CSAE results use validation over factored and unfactored models with sigmoid hidden units. RBF-kernel SVM and RBM results taken from [40]; deep net and GSM results from [30].

be consistent with [30], we furthermore normalized filters to have constant norm during the optimization. For the GSM model, we report the best performance of factored vs. unfactored on the test data, which may introduce a bias in favor of that model. Some example images with corresponding filters learned by the ordinary and factored CSAE model are shown in Figure 9.

5 CONCLUSION

We showed how we may assign an unnormalized energy surface to an autoencoder by interpreting it as a dynamical system. Unlike previous approaches to defining an autoencoder energy, the dynamical systems perspective is not restricted to sigmoid activation functions, which make the autoencoder resemble an RBM, and it is independent of the training criterion.

We also show how multiple class-specific autoencoders can be turned into a generative classifier that yields competitive performance in difficult benchmark tasks. Class-specific dynamical systems may offer an appealing alternative perspective onto classification than the commonly used linear (eg. logistic regression-) layer atop a deep neural network. If a class is represented not just by a weight vector, but by a dynamic sub-network such as an autoencoder, it is easy to model highly complex intra-class variability using a comparably small amount of computational resources.

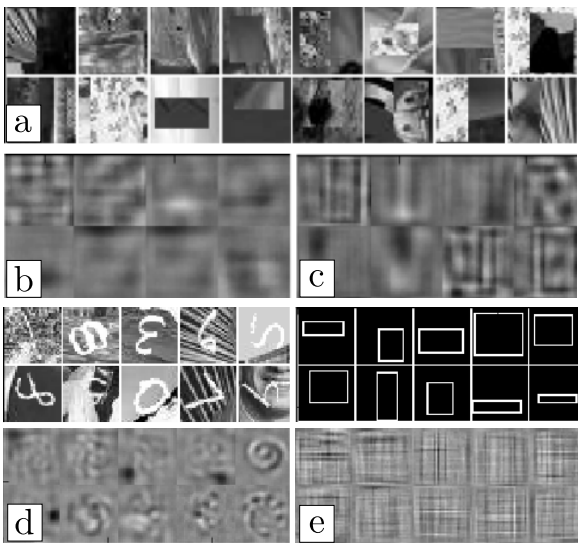

Fig. 9. Example images and filters learned by the CSAE model. **(a)**: Examples of RECTANGLESimg data; **(b)-(c)**: learned horizontal vs vertical filters. **(d)**: MNISTrotImg (factored model); **(e)**: RECTANGLES (factored model).

APPENDIX

A numerically stable computation of $\log(\cosh(u_k))$ may be derived as follows:

$$\begin{aligned}\log(\cosh(u_k)) &= \log\left(\frac{\exp(u_k)+\exp(-u_k)}{2}\right)\\ &= \log(\exp(u_k)+\exp(-u_k)) - \log 2\\ &\propto \log(\exp(u_k)(1+\exp(-2u_k)))\\ &= \log(\exp(u_k)) + \log(1+\exp(-2u_k))\\ &= u_k + \mathrm{softplus}(-2u_k)\end{aligned} \tag{51}$$

REFERENCES

[1] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.
[2] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[3] Y. LeCun, S. Chopra, R. Hadsell, M. A. Ranzato, and F. J. Huang. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press, 2006.

[4] Q. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning (ICML)*, 2012.

[5] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2011.

[6] J. T. Rolfe and Y. LeCun. Discriminative recurrent sparse autoencoders. In *International Conference on Learning Representations (ICLR)*, 2013.

[7] K. Swersky, D. Buchman, B.M. Marlin, and N. de Freitas. On autoencoders and score matching for energy based models. In *International Conference on Machine Learning (ICML)*, 2011.

[8] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, July 2011.

[9] R. Memisevic. Gradient-based learning of higher-order image features. In *the International Conference on Computer Vision (ICCV)*, 2011.

[10] W.Y. Zou, S. Zhu, A. Ng, and K. Yu. Deep learning of invariant features via simulated fixations in video. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[11] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, December 2005.

[12] Aapo Hyvarinen. Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables. *Neural Networks, IEEE Transactions on*, 18(5):1529–1531, 2007.

[13] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *International Conference on Machine Learning (ICML)*, 2011.

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

[15] H.S. Seung. Learning continuous attractors in recurrent networks. *Advances in neural information processing systems (NIPS)*, 10:654–660, 1998.

[16] Hanna Kamyshanska and Roland Memisevic. On autoencoder scoring. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 720–728, 2013.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008.

[18] G. Alain, Y. Bengio, and S. Rifai. Regularized auto-encoders estimate local statistics. *arXiv preprint arXiv:1211.4246*, 2012.

[19] John M Lee. Introduction to smooth manifolds, 2001.

[20] RM Santilli. Foundations of theoretical mechanics II. Birkhoffian generalization of Hamiltonian mechanics. 1982.

[21] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. *Advances in neural information processing systems (NIPS)*, 17, 2005.

[22] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[23] E.H. Adelson and J.R. Bergen. Spatiotemporal energy models for the perception of motion. *J. Opt. Soc. Am. A*, 2(2):284–299, 1985.

[24] Roland Memisevic. Learning to relate images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1829–1846, 2013.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Frnkranz and Thorsten Joachims, editors, *ICML*, 2010.

[26] Kishore Reddy Konda, Roland Memisevic, and Vincent Michalski. The role of spatio-temporal synchrony in the encoding of motion. *arXiv preprint arXiv:1306.3162*, 2013.

[27] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[28] TM Cover and J Thomas. *Elements of Information Theory*. New York: John Wiley & Sons, Inc, 1991.

[29] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[30] R. Memisevic, C. Zach, G. Hinton, and M. Pollefeys. Gated softmax classification. *Advances in Neural Information Processing Systems (NIPS)*, 23, 2011.

[31] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning (ICML)*, 2007.

[32] H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2008.

[33] Tanya Schmah, Geoffrey E Hinton, Steven L Small, Stephen Strother, and Richard S Zemel. Generative versus discriminative training of rbms for classification of fmri images. In *Advances in neural information processing systems*, pages 1409–1416, 2008.

[34] M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, March 2012.

[35] Graham Taylor and Geoffrey Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[36] Roland Memisevic and Geoffrey E Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22(6):1473–92, 2010.

[37] Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits, 1998.

[38] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Python for Scientic Computing Conference (SciPy)*, 2010.

[39] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[40] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

Hanna Kamyshanska studied Mathematics at the National Technical University of Ukraine, Kiew, and received the Diploma in Computer Science from the Goethe University of Frankfurt, Germany, in 2013. She is currently a PhD candidate in Computational Neuroscience in Matthias Kaschube's Group at Frankfurt Institute for Advanced Studies. Her research interests are in computer vision and in the models for visual cortical development in biological systems.

Roland Memisevic received the PhD in Computer Science from the University of Toronto in 2008. Subsequently, he held positions as a research scientist at PNYLab LLC in Princeton, as post-doctoral fellow at the University of Toronto and ETH Zurich, and as a junior professor at the University of Frankfurt, Germany. In 2012, he joined the University of Montreal, Canada, as an assistant professor in Computer Science. His research interests are in machine learning and computer vision.