Neural Networks 64 (2015) 4–11

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2015 Special Issue

Two-layer contractive encodings for learning stable nonlinear features

Hannes Schulz [a,*], Kyunghyun Cho [b], Tapani Raiko [b], Sven Behnke [a]

[a] Autonomous Intelligent Systems, Computer Science Institute VI, University of Bonn, Germany
[b] Department of Information and Computer Science, Aalto University School of Science, Finland

ARTICLE INFO

Article history:
Available online 28 September 2014

Keywords:
Deep learning
Multi-layer perceptron
Two-layer contractive encoding
Linear transformation
Pretraining
Semi-supervised learning

ABSTRACT

Unsupervised learning of feature hierarchies is often a good strategy to initialize deep architectures for supervised learning. Most existing deep learning methods build these feature hierarchies layer by layer in a greedy fashion using either auto-encoders or restricted Boltzmann machines. Both yield encoders which compute linear projections of input followed by a smooth thresholding function. In this work, we demonstrate that these encoders fail to find stable features when the required computation is in the exclusive-or class. To overcome this limitation, we propose a two-layer encoder which is less restricted in the type of features it can learn. The proposed encoder is regularized by an extension of previous work on contractive regularization. This proposed two-layer contractive encoder potentially poses a more difficult optimization problem, and we further propose to linearly transform hidden neurons of the encoder to make learning easier. We demonstrate the advantages of the two-layer encoders qualitatively on artificially constructed datasets as well as commonly used benchmark datasets. We also conduct experiments on a semi-supervised learning task and show the benefits of the proposed two-layer encoders trained with the linear transformation of perceptrons.

1. Introduction

Unsupervised learning of feature hierarchies – pre-training – has often been used in recent years for the initialization of supervised learning of deep architectures, e.g. for the categorization of images. This initialization was found to improve results for many-layered – so-called *deep* – neural networks (Bengio, Lamblin, Popovici, & Larochelle, 2007; Hinton & Salakhutdinov, 2006; Ranzato, Poultney, Chopra, & LeCun, 2007) and has spurred research on understanding and improving feature learning methods (e.g. Cho, Raiko, & Ilin, 2011; Erhan, Manzagol, Bengio, Bengio, & Vincent, 2009; Glorot & Bengio, 2010a; Rifai, Vincent, Muller, Glorot, & Bengio, 2011c). Classical pre-training for a multi-layer perceptron is performed layer-wise in a greedy fashion, that is, after training the parameters of one layer, they are fixed for the training of the next higher layer parameters. After pre-training, all parameters are *fine-tuned* jointly in a supervised manner on the task.

The greedy initialization steps successively build more complex features and at the same time avoid problems occurring when training deep architectures directly. Erhan et al. (2009) argue that many-layered architectures have more local minima and that gradients are becoming less informative when passing through many layers. In contrast, commonly employed auto-encoders (AE) and restricted Boltzmann machines (RBM) are shallow. They have fewer local minima and gradients are not diluted.

In some cases, layer-wise pre-training might not help fine-tuning, e.g. when extracted features bear no relation with the desired output. Recently, Rifai et al. (2011c) showed that *stable* features of the training data are useful for supervised training on a wide range of datasets. These features do not change when the input varies slightly. The failure mode we address in this paper is when these stable features cannot be recognized by the pre-training method. AEs and RBMs both yield encoders which consist of a linear projection followed by a smooth thresholding function. This is a highly restricted function class. In fact, while the deep MLP can learn almost arbitrary functions (Hornik, Stinchcombe, & White, 1989), Minsky and Seymour (1969) showed that a one-layer neural network as is used for AE and RBM is not able to learn the class of not linearly separable functions, of which the well-known XOR problem is the simplest example.

We argue here that deep architectures pre-trained with the common one-layer encoders often fail to discover features which

[*] Corresponding author. Tel.: +49 228734175.
E-mail address: schulz@ais.uni-bonn.de (H. Schulz).

http://dx.doi.org/10.1016/j.neunet.2014.09.008
0893-6080/© 2014 Elsevier Ltd. All rights reserved.

are of the XOR class (which we shall refer to as *non-linear* features), and that fine-tuning may not repair this defect. We construct problems that cannot profit from pre-training and show that pre-training may even be counter-productive in these cases.

These problem cases can be solved for auto-encoders by introducing a hidden layer in the encoder, yielding a compromise between the advantages of increased expressiveness and disadvantages of increased depth.

To remedy the problem of increased depth, we propose to extend contractive regularization (Rifai et al., 2011c) to two-layer auto-encoder pre-training. We further propose to add shortcuts to the autoencoder, which helps learning a combination of simple and complex features. Our training procedure employs the method of linearly transforming perceptrons, recently proposed by Raiko, Valpola, and LeCun (2012).

We show that contractive regularization can resolve the constructed cases and performs better than greedy pre-training on benchmark datasets. Finally, we evaluate the proposed two-layer encoding with shortcut method on the task of semi-supervised classification of handwritten digits and show that it achieves better generalization than greedy pre-training methods when only a few labeled examples are available.

2. Related work

The representational power of deep architectures has been thoroughly analyzed (Bengio & Delalleau, 2011; Le Roux & Bengio, 2008). Le Roux and Bengio (2008) showed that, in principle, any distribution can be represented by an RBM with $M + 1$ hidden units, where M is the number of input states with non-zero probability. The question of which *features* can be represented, is not addressed, however. Bengio and Delalleau (2011) analyzed the representational power of deep architectures and showed that they can represent some functions with exponentially less units than shallow architectures. It is not clear, however, whether these representations can be learned in a greedy way.

There is considerable evidence that the performance of deep architectures can be improved when the greedy initialization procedure is relaxed. Salakhutdinov and Hinton (2009) report advantages when performing simultaneous unsupervised optimization of all layer parameters of a stack of RBMs as a deep Boltzmann machine. The authors rely on a greedy initialization, however, which we demonstrate here might establish a bad starting point. Ngiam, Chen, Koh, and Ng (2011) train a deep belief network without greedy initialization and also report good results. Their approach might not scale to many-leveled hierarchies though, and relies on a variant of contrastive divergence to approximate the gradient. Recently, Cireşan, Meier, and Schmidhuber (2012) obtained top results on a number of image classification datasets with deep neural networks. They did not rely on pre-training, but used other regularization methods, such as convolutional network structure with max-pooling, generation of additional training examples trough transformations, and bagging.

3. Background

In this section, we discuss each of the three methods combined on our approach – auto-encoders, contractive encodings and linear transformations – in more detail.

3.1. Auto-encoders

An auto-encoder consists of an encoder and a decoder. The encoder typically has the form

$$\mathbf{h} = \mathbf{f}_{\text{enc}}(\mathbf{x}) = \boldsymbol{\sigma}(\mathbf{o}) = \boldsymbol{\sigma}(\mathbf{A}\mathbf{x}), \tag{1}$$

where $\boldsymbol{\sigma}$ is a component-wise sigmoid non-linearity, e.g. $\sigma_i(\mathbf{o}) = \tanh(o_i)$. The encoder transforms the input $\mathbf{x} \in \mathbb{R}^N$ to a hidden representation $\mathbf{h} \in \mathbb{R}^M$ via the (learned) matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$. The decoder is then given by

$$\hat{\mathbf{x}} = \mathbf{f}_{\text{dec}}(\mathbf{h}) = \mathbf{A}'\mathbf{h} \in \mathbb{R}^N, \tag{2}$$

where we restrict ourselves to the symmetric case $\mathbf{A}' = \mathbf{A}^\top$. Even though biases are commonly used, we omit them here for clarity. The main objective for auto-encoders is to determine $\mathbf{A}$ such that $\hat{\mathbf{x}}$ is similar to $\mathbf{x}$. For binary $\mathbf{x}$, this amounts to minimizing the cross-entropy loss

$$\ell_{\text{bin}}(\mathbf{x}, \mathbf{A}) = -\sum_i^N \left(x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)\right). \tag{3}$$

Auto-encoders have gained popularity as a method for pre-training of deep neural networks (e.g. Bengio, Lamblin, Popovici, & Larochelle, 2006). In deep neural networks, gradient information close to the input layer is “diluted” since it passed through a series of randomly initialized nonlinear layers. This effect makes it difficult to learn good high-level representations (Erhan et al., 2010). Pre-training moves weights to an area where they relate to the input and therefore allow for cleaner gradient propagation. Bengio (2009) and Bengio, Courville, and Vincent (2013) further hypothesize that stacking of unsupervised neural networks disentangles factors of variations and that the untangled representations make supervised learning easier.

Pre-training typically proceeds in a greedy fashion. Consider a deep network for a classification task, $\mathbf{y} = \mathbf{A}^{(L)}\mathbf{h}^{(L)}$, with $\mathbf{h}^{(l+1)} = \boldsymbol{\sigma}(\mathbf{A}^{(l)}\mathbf{h}^{(l)})$, where $\mathbf{y}$ is the output layer, $\mathbf{h}^{(l)},\ l \in 1, \dots, L$ are hidden layers, and $\mathbf{h}^{(0)} = \mathbf{x}$ is the input layer. Then, $L - 1$ auto-encoders are learned in succession, minimizing $\ell_{\cdot}(\mathbf{h}^{(l)}, \mathbf{A}^{(l)})$ and keeping $\mathbf{A}^{(l'<l)}$ fixed. It is frequently observed that following this protocol, successively higher-level representations of the inputs are attained (Bengio et al., 2006). This pre-training is, however, greedy, and a joint optimization of all layers might yield superior results in principle. In practice, however, joint optimization without pre-training suffers from the same gradient dilution problem as originally addressed by the deep-learning methodology and often yields bad performance. After pre-training, the supervised classification objective is optimized jointly with respect to all parameters $(\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(L)})$. This final step is called *fine-tuning*.

3.2. Contractive encodings

To avoid overfitting the training data, or in order to obtain the overcomplete representations, it is common to regularize the auto-encoder learning. A common regularizer for MLPs is the L_2 penalty on the weight matrices. This regularizer is well-motivated for linear methods (e.g. ridge regression or logistic regression), where it penalizes strong dependence of $\mathbf{y}$ on few variables in $\mathbf{x}$, and thus ensures invariance of $\mathbf{y}$ to small changes in $\mathbf{x}$.

For MLPs, which contain saturating non-linearities, this desirable property can be achieved with both strongly positive and negative weights. Rifai et al. (2011a) and Rifai, Vincent, Muller, Glorot, and Bengio (2011d) show that the generalization of the L_2 penalty in the presence of saturating non-linearities is the contractive penalty.

The contractive penalty penalizes the squared Frobenius norm of the Jacobian $J_{\mathbf{f}_{\text{enc}}}$ of $\mathbf{f}_{\text{enc}}$ with respect to the input $\mathbf{x}$, where the Jacobian is defined by

$$J_{\mathbf{f}_{\text{enc}}} = \begin{bmatrix} \dfrac{\partial f^1_{\text{enc}}}{\partial x_1} & \cdots & \dfrac{\partial f^1_{\text{enc}}}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f^M_{\text{enc}}}{\partial x_1} & \cdots & \dfrac{\partial f^M_{\text{enc}}}{\partial x_N} \end{bmatrix}.$$

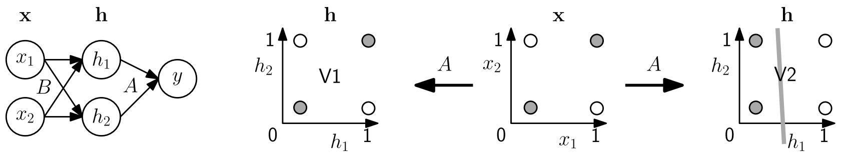

Fig. 1. Auto-encoder pre-training can be counter-productive. The simple network on the left should learn $y = x_1 \veebar x_2$. Pre-training of A makes an uninformed choice between representations V1 and V2 without loss of generality, but only V2 is linearly separable and helps fine-tuning.

By minimizing the Jacobin $J_{\mathbf{f}_{\text{enc}}}$, the hidden representation computed by the encoder becomes more invariant to small changes in the input $\mathbf{x}$, resulting in robust representations.

The combined objective is given as

$$\ell_{CAE}(\mathbf{h}^{(l)}, \mathbf{A}^{(l)}) = \ell_{\cdot}(\mathbf{h}^{(l)}, \mathbf{A}^{(l)}) + \lambda \|J_{\mathbf{f}_{\text{enc}}}\|^2, \quad (4)$$

where λ is the regularization strength. Rifai et al. (2011a, 2011d) demonstrate that training a stack of simple auto-encoders with the contractive penalty produces features which identify the data manifold, which later on helps fine-tuning.

3.3. *Linear transformations in perceptrons*

Raiko et al. (2012) proposed a method of linearly transforming perceptrons in a deep MLP network to avoid difficulties in training a deep neural network without pre-training.

Let us focus on a single hidden layer within a possibly deep MLP network. The inputs to this layer are denoted $\mathbf{x}^k$ and its outputs are $\mathbf{y}^k$, where k is the sample index. We allow shortcut connections that by-pass one or more hidden layers such that the inputs to each hidden layer may be distributed over several previous layers of the network. The mapping from $\mathbf{x}^k$ to $\mathbf{y}^k$ is modeled as

$$\mathbf{y}^k = \mathbf{A}\sigma\left(\mathbf{B}\mathbf{x}^k\right) + \mathbf{C}\mathbf{x}^k, \quad (5)$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are weight matrices. In order to avoid separate bias vectors that complicate formulas, the input vectors $\mathbf{x}^k$ are assumed to have been supplemented with an additional component that is always one.

Let us assume that σ is a tanh nonlinearity and supplement it with auxiliary scalar variables α_i and β_i for each component σ_i. We define

$$\sigma_i(\mathbf{b}_i\mathbf{x}^k) = \tanh(\mathbf{b}_i\mathbf{x}^k) + \alpha_i\mathbf{b}_i\mathbf{x}^k + \beta_i, \quad (6)$$

where $\mathbf{b}_i$ is the ith row vector of matrix $\mathbf{B}$.

α_i's and β_i's are updated during training in order to help learning the other parameters $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. By updating α_i's and β_i's we will ensure that

$$0 = \sum_{k=1}^{K} \sigma_i(\mathbf{b}_i\mathbf{x}^k), \quad \text{and} \quad 0 = \sum_{k=1}^{K} \sigma_i'(\mathbf{b}_i\mathbf{x}^k). \quad (7)$$

These are satisfied by setting α_i and β_i to

$$\alpha_i = -\frac{1}{K}\sum_{k=1}^{K} \tanh'(\mathbf{b}_i\mathbf{x}^k),$$

$$\beta_i = -\frac{1}{K}\sum_{k=1}^{K} \left[\tanh(\mathbf{b}_i\mathbf{x}^k) + \alpha_i\mathbf{b}_i\mathbf{x}^k\right].$$

We motivate these seemingly arbitrary update rules below.

The effect of changing the transformation parameters α_i and β_i are compensated exactly by updating the shortcut mapping $\mathbf{C}$ by

$$\mathbf{C}_{\text{new}} = \mathbf{C}_{\text{old}} - \mathbf{A}(\boldsymbol{\alpha}_{\text{new}} - \boldsymbol{\alpha}_{\text{old}})\mathbf{B} - \mathbf{A}(\boldsymbol{\beta}_{\text{new}} - \boldsymbol{\beta}_{\text{old}})\,[0\,0\cdots 1]\,, \quad (8)$$

where $\boldsymbol{\alpha}$ is a matrix with elements α_i on the diagonal and one empty row below for the bias term, and $\boldsymbol{\beta}$ is a column vector with components β_i and one zero below for the bias term. Thus, any change in α_i and β_i does not change the overall mapping from $\mathbf{x}^k$ to $\mathbf{y}^k$ at all, but they do change the optimization problem instead.

One way to motivate the transformations in Eq. (7), is to study the expected output $\mathbf{y}_t$ and its dependency on the input $\mathbf{x}_t$:

$$\frac{1}{K}\sum_k \mathbf{y}^k = \mathbf{A}\left[\frac{1}{T}\sum_k \boldsymbol{\sigma}(\mathbf{B}\mathbf{x}^k)\right] + \mathbf{C}\left[\frac{1}{K}\sum_k \mathbf{x}^k\right], \quad (9)$$

$$\frac{1}{K}\sum_k \frac{\partial \mathbf{y}^k}{\partial \mathbf{x}^k} = \mathbf{A}\left[\frac{1}{K}\sum_k \boldsymbol{\sigma}'(\mathbf{B}\mathbf{x}^k)\right]\mathbf{B}^\top + \mathbf{C}. \quad (10)$$

We note that by making nonlinear activations $\boldsymbol{\sigma}(\cdot)$ zero mean in Eq. (7) (left), we disallow the nonlinear mapping $\mathbf{A}\boldsymbol{\sigma}(\mathbf{B}\cdot)$ to affect the expected output $\mathbf{y}^k$, that is, to compete with the bias term. Similarly, by making the nonlinear activations $\boldsymbol{\sigma}(\cdot)$ zero slope in Eq. (7) (right), we disallow the nonlinear mapping $\mathbf{Af}(\mathbf{B}\cdot)$ from affecting the expected dependency on the input, that is, to compete with the linear short-cut mapping $\mathbf{C}$. In traditional neural networks, the linear dependencies (expected $\partial\mathbf{y}^k/\partial\mathbf{x}^k$) are modeled by many competing paths from an input to an output (via each hidden unit), whereas this architecture gathers the linear dependencies to be modeled only by $\mathbf{C}$.

Raiko et al. (2012) showed experimentally that less competition between parts of the model speeds up learning significantly. It also helped getting state-of-the-art learning results for MLP networks on three tasks (MNIST classification, CIFAR-10 classification, and MNIST deep auto-encoders). Vatanen, Raiko, Valpola, and LeCun (2013) drew more careful connections to second-order optimization methods, showing that the reparameterization done using transformations make first-order optimization methods behave more like a second-order method.

Recently, Dauphin et al. (2014) argued that plateaus around saddle points in the parameter space dramatically slow down learning and giving an illusory impression of local minima. They proposed a second-order optimization method that is able to escape saddle points and showed that one can continue optimization from a seemingly converged optimization by a first-order method. They also discussed whether actual local minima are as big an issue as has long been thought. It remains an interesting open issue whether the good empirical performance of the transformations is related to this phenomenon.

4. Where pre-training of one-layer encoders fails

Let us assume that we want to approximate the Boolean function $f(x_1, x_2) := x_1 \veebar x_2$, where $\cdot \veebar \cdot$ denotes the exclusive-or relation (XOR). For this purpose, we consider the neural network with a two-unit hidden layer shown in Fig. 1 (left) and perform auto-encoder pre-training for the first-layer matrix B. During the pre-training phase, two filters have to be learned, mapping the input vector again to a two-dimensional space $\mathbf{h} = (h_1, h_2)$. Without further information, this mapping might choose a representation which is not linearly separable (denoted "V1" in Fig. 1). In this case, pre-training does not aid fine-tuning, it chooses a feature representation that is not helpful for the classification task. Of course, this argument merely stresses the distinction between supervised and unsupervised learning. It does not follow that pre-training is not helpful *per se*. Our observation has, however, important consequences for *greedy* pre-training.

We can easily extend the argument of the previous paragraph to a case where greedy pre-training does not find stable non-linear features that are obvious from the data. Let us assume we have a dataset of three variables, where $\mathbf{x}^k = (x_1^k, x_2^k, x_1^k \veebar x_2^k)$. The only stable feature of this dataset is $x_1 \veebar x_2$, i.e. a two-layered denoising auto-encoder should be able to recover x_3^k from the first two components and any of the x_1^k, x_2^k from the other variable and $x_1^k \veebar x_2^k$. If unfortunate greedy training of the first layer prevents the second layer from learning that x_1 and x_2 are XOR-related – as demonstrated above – the second layer will fail to discover this relation. Even worse, pre-training might leave the weights in a state where recovery using fine-tuning is not possible. We will empirically verify these claims in the experiments section.

4.1. Two-layer encoders and contractive regularization

Considering the failure mode of one-layer encoders discussed above, an intuitive extension is to make the encoder more powerful by adding a hidden layer $\mathbf{h}' \in \mathbb{R}^P$ to the encoder, such that $\mathbf{f}_{\text{enc}}(\mathbf{x}) = \mathbf{h} = \sigma(\mathbf{A}\mathbf{h}') = \sigma(\mathbf{A}\sigma(\mathbf{B}\mathbf{x}))$, with $\mathbf{x} \in \mathbb{R}^M$, $\mathbf{A} \in \mathbb{R}^{N\times P}$, $\mathbf{B} \in \mathbb{R}^{P\times M}$. Extending contractive regularization (Rifai et al., 2011c) to two-layer encoders yields

$$\|J_{\mathbf{f}_{\text{enc}}}(\mathbf{x})\|_F^2 = \sum_n^N \sum_m^M \left(1 - \mathbf{f}_{\text{enc}}(\mathbf{x})_n^2\right)^2 \times \left(\sum_p^P \mathbf{A}_{np}\mathbf{B}_{pm}(1 - \mathbf{h}(\mathbf{x})_p^2)\right)^2, \tag{11}$$

where $\mathbf{h}(\mathbf{x})$ is the hidden layer activation. In contrast to one-layer contractive regularization, which has the complexity of a forward pass, this regularizer is $O(MPN)$. To reduce the time required to do experiments, we only compute the regularizer for few randomly chosen instances in the mini-batch. The precise number is a tradeoff between acceptable training time and accuracy, which needs to be cross-validated. The simultaneous perturbation method (SPSA, Spall, 1998), which has not received much attention recently, could potentially be employed to reduce the regularization cost to a second forward-pass.

4.2. Two-layer encoders and shortcut connections

In this paper, we argue that while two-layer encodings are harder to learn, we can combine their ability to detect highly non-linear features with the easy-to-learn one-layer encodings by introducing shortcuts. Shortcut weights $\mathbf{C}$ (see, e.g., Eq. (5)) from the input to the second hidden layer can be regularized as in Rifai et al. (2011d), while the two-layer encoder is regularized as introduced above. We employ linear transformations and compensations (Section 3.3) to ensure that simple features continue to be learned by the shortcut weights, while the two-layer part of the encoder can focus on the difficult features. For this purpose, we extend the two-layer contractive regularizer to account for the linear transformations in (5) and (6),

$$\left\|J_{\mathbf{f}_{\text{enc}}}(\mathbf{x})\right\|_F^2 = \sum (1 - \mathbf{f}_{\text{enc}}(\mathbf{x})^2)^{2\top} \left(\mathbf{C} + \mathbf{A}(\mathbf{B}^\alpha + \mathbf{B}')\right)^2, \tag{12}$$

where $\mathbf{B}^\alpha_{pm} = \alpha_p \mathbf{B}_{pm}$, $\mathbf{B}'_{pm} = \mathbf{B}_{pm}(1 - \tanh^2(\mathbf{B}_p.\mathbf{x}))$. Fig. 2 illustrates the proposed encoder structure.

5. Experiments

In our experiments, we aim to establish the following claims:

1. One-layer encoders are severely restricted.
2. Two-layer encoders can be better learned with the proposed contractive regularizer.
3. Two-layer encoders can further profit from shortcut connections in the case of semi-supervised learning.

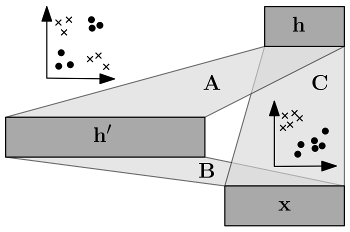

Fig. 2. Schematic visualization of our encoder. Features **h** of input **x** are determined both by a one-layer encoder via **C**, and by a two-layer encoder via **B** and **A**. Contractive regularization (Rifai et al., 2011d) and two-layer contractive regularization (Schulz & Behnke, 2012) are used to learn stable linear/non-linear representations in **h**, respectively. Linear transformations in the two-layer part are moved to **C** using compensations (Raiko et al., 2012) (not shown).

Table 1
Hyper-parameter distribution used in our experiments.

Hyper-parameter	Distribution
Auto-encoder learn rate	$\log \mathcal{U}(0.01, 0.2)$
MLP learn rate	$\log \mathcal{U}(0.01, 0.2)$
Regularization strength (λ)	$\mathcal{U}(0.001, 2)$

Our experiments follow a common protocol. For a fixed architecture, we repeatedly sample all free hyper-parameters from the distributions detailed in Table 1. For a weight matrix $A \in \mathbb{R}^{N\times M}$, weights are initialized uniformly with $a_{ij} \sim \mathcal{U}\big(-\sqrt{6/N+M}, \sqrt{6/N+M}\big)$ as proposed by Glorot and Bengio (2010a). The dataset is split in training, validation and testing sets. We stop each training stage before the loss on the validation set increases. The model with the best final validation error is trained again using training and validation set, for the same number of epochs as determined in the validation phase, and is finally evaluated on the test set.

5.1. Detecting constraint violations in LDPC codes

We now extend the toy example of Section 4 to a more realistic, albeit still constructed, task. A low density parity check (LDPC) code, also known as Gallager code (Gallager, 1962), is a code that allows error correction after transmission through a noisy channel. This is achieved by relating the bits in the message with a set of random constraints known to both sender and receiver. A constraint c over a set of variables $\mathcal{C}$ is met iff $0 \equiv (\sum_{x\in\mathcal{C}} x) \bmod 2$. Note, that the modulo operation generalizes the XOR operation to multiple binary variables.

We consider a subproblem of decoding an LDPC code, namely detecting constraint violations in the code. To this end, we construct a dataset where a code word $\mathbf{w} \in \{0, 1\}^N$ is constrained by N constraints $\mathcal{C}_n$ with three participating variables, each. Each variable participates in three random constraints. Whether a constraint is violated in $\mathbf{w}$ can be determined by a two-layer neural network with $\mathbf{w}$ as its input. The hidden layer has four neurons for every constraint. Each of these four neurons detects a different case where the corresponding three neurons in the input sum to an even number (i.e., for configurations 000, 011, 110, and 101). We denote the weight matrix realizing this $A_{\mathbf{and}}$, as detects conjunctions in the input data. A second weight matrix, $A_{\mathbf{or}}$, then detects whether any of the four neurons in the hidden layer was active and turns on the corresponding neuron in the output layer $\mathbf{v}$. Thus, the values in $\mathbf{v}$ indicate which constraints in $\mathcal{C}$ are violated in $\mathbf{w}$.

For our small problem size, the matrix $\mathcal{C}$ can be generated by creating binary matrices with the correct number of ones and verifying the number of ones per row and column. The matrices $A_{\mathbf{and}}$ and $A_{\mathbf{or}}$ can then be derived directly from the constraint

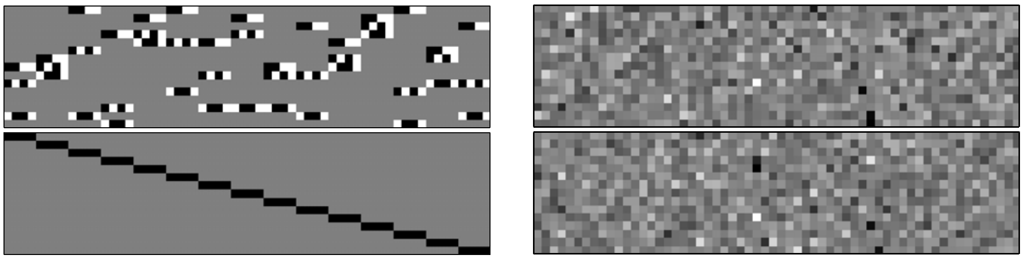

Fig. 3. *Left*: The matrices $A_{\mathbf{and}}$ and $A_{\mathbf{or}}^{\top}$ used in the LDPC experiments. Weights depicted in black, gray, and white denote −1, 0, and 1, respectively. *Right*: Weights of the best-performing auto-encoder after greedy pre-training on LDPC. Sections of first-layer weight matrix roughly corresponding to $A_{\mathbf{and}}$ and $A_{\mathbf{or}}^{\top}$ on the left. Left and right matrices should be equal up to permutation, but the greedy pre-training is unable to detect the non-linear features of the LDPC dataset.

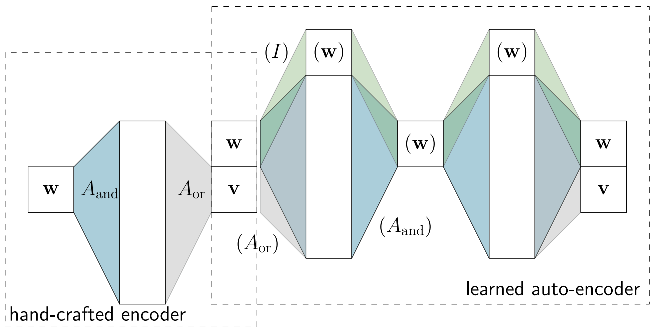

Fig. 4. Constructed example where greedy auto-encoders fail. The matrices $A_{\mathbf{and}}$ and $A_{\mathbf{or}}$ are hand-crafted to calculate $\sum_i w_i \mod 2$. Matrices and resulting representations in parenthesis have to be recovered, i.e. learned, by the auto-encoder to solve the reconstruction task.

matrix. The matrices used in our experiments are shown on the left side of Fig. 3.

We now frame learning $A_{\mathbf{and}}$ and $A_{\mathbf{or}}$ as a task for the auto-encoder shown in Fig. 4. The auto-encoder reconstructs the code word $\mathbf{w}$ and the constraint violation vector $\mathbf{v}$ together:

$$(w_1, w_2, \dots, w_n, v_1, v_2, \dots, v_n) = \mathbf{x} \overset{!}{=} \mathbf{f}_{\text{dec}}(\mathbf{f}_{\text{enc}}(\mathbf{x})), \tag{13}$$

$$\text{where } \mathbf{f}_{\text{enc}}(\mathbf{x}) = \boldsymbol{\sigma}(A_{\mathbf{or}}\,\boldsymbol{\sigma}(A_{\mathbf{and}}\,\mathbf{x})) \tag{14}$$

$$\text{and} \quad \mathbf{f}_{\text{dec}}(\mathbf{h}) = \boldsymbol{\sigma}(A_{\mathbf{and}}^{\top}\,\boldsymbol{\sigma}(A_{\mathbf{or}}^{\top}(\mathbf{h}))). \tag{15}$$

Since $w_i \sim \text{Binomial}(1, 0.5)$, $\mathbf{w}$ cannot be compressed to less than N bits. Hence, a hidden layer size N creates a bottleneck where no more than the complete codeword can be represented.

The dataset is constructed from all $N = 15$ bit strings and randomly split into training (60%), validation (20%), and test set (20%). After pre-training, we fine-tune all weight matrices of the network by reconstructing $(\mathbf{w}, \mathbf{v})$ using the logistic reconstruction loss—again with early stopping on the validation set. An instance is reconstructed correctly if the sign of the outputs corresponds to the input. This task is clearly related to the pre-training objective, as during pre-training the hidden layer should fully represent the code.

The results are visualized in Fig. 5(a). We show the fraction of draws from the hyper-parameters which performs better than a given validation error. For greedy pre-training, the chances of finding a model which performs well on the validation set are very small. This is reflected in the test errors displayed in Table 2 (LDPC). Only conditions where both layers are trained *simultaneously* are able to solve the task. The problem is also apparent in the learned weights after pre-training, shown on the right side of Fig. 3. Since the relations in the dataset can only be detected in the second hidden layer, greedy pre-training cannot learn anything useful. Finally, we compared the models to a model where we removed the first hidden layer. As expected from the construction of the dataset, this network architecture is insufficient to solve the task and yields the highest error.

Table 2
Comparison of pre-training methods for fixed architecture.

Condition			Test error (%)		
# Layers	Pre-training	Regularization	LDPC	MNIST-rot	MNIST
1	No	None	88.6	13.8	1.8
1	Yes	None	81.8	15.7	1.6
1	Yes	Contractive	71.1	14.1	1.6
2	Greedy	Contractive	6.8	13.2	1.6
2	Greedy	None	5.3	14.6	1.7
2	No	None	**0.0**	12.5	1.7
2	Non-greedy	None	**0.0**	12.5	1.7
2	Non-greedy	Contractive	**0.0**	**11.4**	**1.4**

The model with first and second layer size of 75 and 15, respectively, is the smallest possible model to solve the task. Since pre-training relies on chance, its performance should improve with the layer size. To analyze the difficulty of the task, we increased the number of neurons in the first hidden layer by 400% and found only marginal improvements in the error after pre-training, as shown in Fig. 6. Thus, relying on chance to find good non-linear features does not work well for the LDPC dataset. Increasing the size of the second layer, on the other hand, almost solves the dataset at 160% of the minimal size. This is also expected, since the second layer size mainly influences the total compression ratio of the auto-encoder. In the case of $M = 25$, the ratio is reduced from a factor of $30/15 = 2$ to $30/25 = 1.2$.

Our experiments demonstrate that greedy layer-wise pre-training drives an auto-encoder to learn mainly "linear" features. Non-linear relations contained in the data cannot always be recovered by higher layers, confirming our first claim.

5.2. Benchmark datasets

We also compare our approach on two benchmark datasets, MNIST and the rotated MNIST dataset MNIST-rot. Here, we fix the architecture of the network to input size $N = 784$, first hidden layer size $P = 1000$, second hidden layer size $M = 500$, and choose a batch size of 16. Qualitatively, we get the same results as in the constructed LDPC example for both datasets. The two-layer regularized encoder is more robust with respect to choice of hyper-parameters (Fig. 5(b)) and finds better minima (Table 2, MNIST-rot and MNIST). Additionally, we analyzed the reconstruction error after pre-training models with best classification performance. On the MNIST-rot validation set, the one-layer case achieves an error of 126.3, greedy contractive pre-training yields 98.3, and the regularized two-layer encoder reaches 88.9. The reconstruction results for MNIST have the same ranking. This demonstrates that the features learned in the two-layer encoder are not only better for classification, they are also better representations of the input, which strongly supports our second claim.

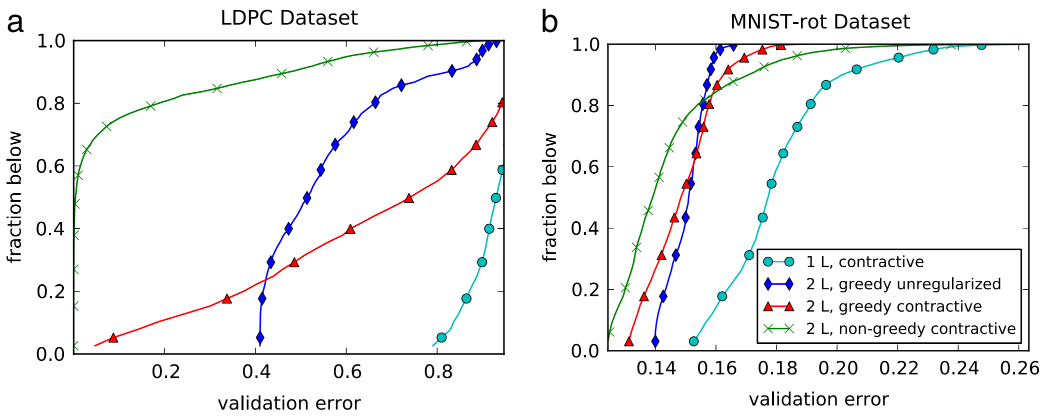

Fig. 5. Comparison of pre-training effects for fixed architecture (30 × 75 × 15 for LDPC, 784 × 1000 × 500 for MNIST-rot). Graphs show fraction of draws from hyper-parameter distribution which performs better than given validation error.

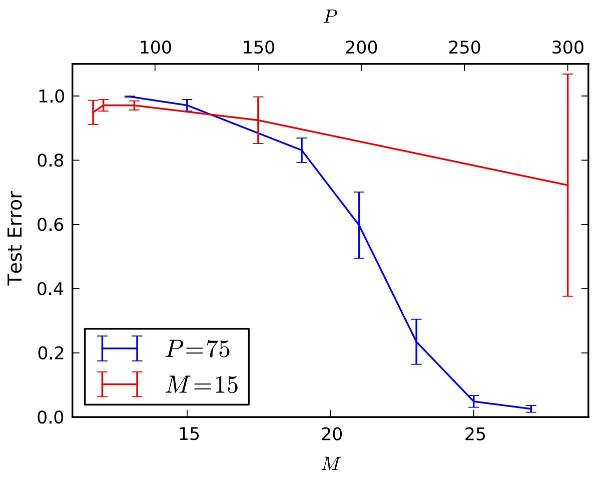

Fig. 6. Performance of models on LDPC dataset after greedy pre-training as a function of layer size. While increasing the second layer size M with fixed $P = 75$ quickly solves the task, increasing the first layer size P while fixing $M = 15$ only helps marginally. Error-bars show standard deviations over five cross-validation runs of the best selected hyper-parameter configuration.

5.3. *Two-layer encoders and shortcut connections*

We evaluate the shortcut connections in a semi-supervised setting on MNIST. We assume that only 1200 training samples have their labels available, while all the other training samples are unlabeled. The task is to use an MLP trained on the training samples to classify 10 000 test samples.

Our base model is a multi-layer perceptron (MLP) with two hidden layers having tanh hidden neurons. The output of the MLP is

$$\mathbf{y} = \mathbf{W}\left(\sigma\left(\mathbf{A}\sigma\left(\mathbf{B}\mathbf{x}\right)\right)\right), \tag{16}$$

where **W**, **A** and **B** are weight matrices. We have omitted biases for simplicity of notation. As baseline, we trained this MLP both with and without pre-training. For the pre-trained MLP, we consider the bottom two layers as an autoencoder with two hidden layers and trained them using both labeled and unlabeled samples.

When the hidden neurons, or perceptrons, in the MLP were linearly transformed, we added shortcut connections from the input to the second hidden layer to maintain the equivalence after the transformation. In that case, the output of the MLP is

$$\mathbf{y} = \mathbf{W}\mathbf{h} = \mathbf{W}\sigma(\mathbf{A}\mathbf{h}' + \mathbf{C}\mathbf{x}), \tag{17}$$

where $\mathbf{h}' = \boldsymbol{\sigma}(\mathbf{B}\mathbf{x}) + \mathbf{B}^{\alpha}\mathbf{x} + \boldsymbol{\beta}$, and **C** is the weight matrix of the shortcuts.[1]

As a comparison, we tried both using either one of the two-layer contractive encoding and the linear transformation and using both of them together. In this way, we can easily see the effectiveness of the proposed way of using both approaches together. Specifically, we used six different training strategies:

1. **S**: MLP trained with labeled samples (S) only,
2. **U + S**: MLP pre-trained with unlabeled samples (U) and fine-tuned (S),
3. **C+U+S**: MLP pre-trained (U) and fine-tuned (S) with two-layer contractive encoding (C),
4. **T + U + S**: MLP with shortcuts pre-trained (U) and fine-tuned (S) with linear transformation (T),
5. **2C + U + S**: MLP pre-trained (U) with stacked contractive auto-encoders (2C) and fine-tuned (S), and
6. **C+T+U+S**: MLP with shortcuts pre-trained (U) and fine-tuned (S) using both the two-layer contractive encoding (C) and linear transformation (T)

We estimated hyper-parameters such as learning rates, weight decay constant, contractive regularization strength and the sizes of the hidden layers using hyperopt (Bergstra, Bardenet, Bengio, & Kégl, 2011). For any unregularized models, we used cross-validated L2 weight decay.

We used five-fold cross-validation, with 48 000 and 12 000 examples for training and validation, respectively, in pre-training. Pre-training was stopped when the loss on the validation set failed to improve. For fine-tuning we used 1000 examples for training and 200 for validation. Due to the small validation set size, we took great care in selecting the best model. We first determine the expected number of training epochs until the optimum is reached by averaging the early-stopping epoch over all folds. The model performance is then given by the average validation set classification error over folds at this epoch during training.

The weight matrices **A** and **B** were initialized randomly according to the normalized scale (Glorot & Bengio, 2010b), while **C** was initialized with zeros.

In Table 3, the resulting classification accuracies for all six strategies are presented. As expected, any approach with pre-training significantly outperforms the case where only labeled samples were used for supervised training (**S**). The best performing strategy was the one which pre-trained the MLP as the two-layer

[1] When we pre-trained the MLP as a two-layer contractive encoding, we tied the weights **A** and **B** between the encoder and decoder. However, we did not share **C**, α_l's and β_l's.

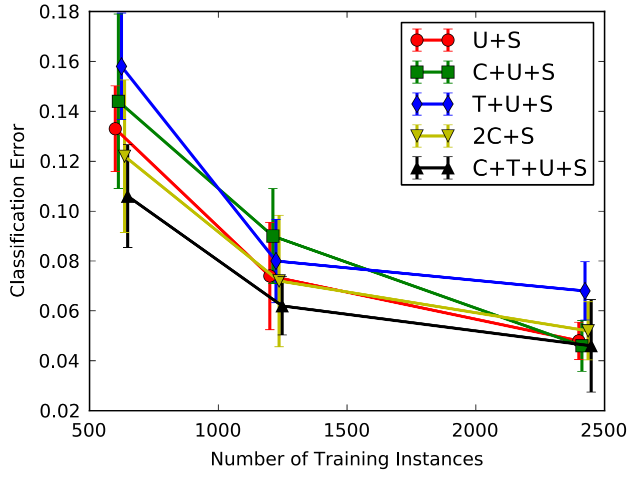

Fig. 7. Test error for varying fine-tuning dataset size (600, 1200, 2400). Standard deviations are over five trials with different draws of the dataset. All fine-tuning hyperparameters were cross-validated for every point.

Table 3
Classification accuracies depending on training strategy on MNIST using 1200 labeled examples. Standard deviations are over five trials with different draws of the training set.

Condition	Test error
S	10.0 ± 1.4
U + S	7.4 ± 2.2
C + U + S	7.8 ± 1.3
T + U + S	8.0 ± 1.7
2C + U + S	7.2 ± 2.6
C + T+ U + S	6.2 ± 1.2

contractive encoding using the linear transformation (**C+T+U+S**). This strategy was able to outperform the strategies **U + S**, **C + U + S** as well as **T + U + S**. Our proposed method also has an advantage over the stacked contractive auto-encoder (**2C + U + S**). This trend also holds when the fine-tuning dataset size is decreased, with the ratio between test and validation set size held constant, as shown in Fig. 7. At larger fine-tuning dataset sizes, the difference between the regularized methods vanishes.

Interestingly, using either the two-layer contractive encoding or the linear transformation only turned out to be only as good as the naïve pre-training strategy (**U + S**). This suggests that it is not easy to train the two-layer contractive encoding well without a good training algorithm. Only when training became easier by linearly transforming perceptrons to have zero-mean and zero-slope on average, we were able to see the improvement (**C + T + U + S**), which confirms our third claim.

6. Discussion

Two-layer encoders are a natural extension of greedy pre-training that enables learning of non-linear features. As stressed in Section 5.1, an important class of features are disjunctions of conjunctions. These functions have a large number of intermediate results compared to the number of outputs, which requires a large number of features in the hidden layers. Another important class of functions are subspace projections, for instance in independent subspace analysis (ISA), where the projection of features in the subspace is stable, but the computed features are not.

Such overcomplete features, where $M > N$, are often successfully used in deep learning, last but not least by Rifai et al. (2011b, 2011c). Intuitively, the overcomplete representation in the first hidden layer provides the second hidden layer more combinations to choose from. A two-layer auto-encoder, on the other hand, does not need to guess which features might help in the higher layer, since they are jointly determined. Therefore, two-layer auto-encoders might be able to achieve higher performance more reliably with the same number of hidden units, which is supported in our experiments by greater robustness to the choice of hyper-parameters and lower reconstruction error.

Especially the LDPC example dataset shows that overcomplete intermediate representations are crucial for some tasks. With one-layer encoders, strong regularization is required to avoid learning the identity function. A two-layer encoder, on the other hand, can have an arbitrarily large hidden layer which only captures the intermediate results of the feature calculation. Regularization – here, stability of the features – is only applied to the second hidden layer, which does not need to be overcomplete.

Along with its advantage, the two-layer encoder comes with a difficulty in training. We observed this difficulty in the case of semi-supervised learning where only a fraction of samples were allowed to have labels. The method of linearly transforming neurons in a deep neural network was used to overcome this difficulty. Interestingly, the experiments showed that it is important to use both the two-layer contractive encoding as well as the linear transformation to achieve good performance.

7. Conclusions

Common pre-training of deep architectures by RBM and AE simplifies one hard deep problem to multiple less difficult single-layer ones. In this paper, we argued that this simplification goes one step too far, to the extent where the class of features which can be learned by the pre-training procedure is restricted severely.

Guided by the observation that one-layer neural networks cannot learn functions in the exclusive-or class, we constructed a task to detect constraint violations in low density parity check codes, which relies heavily on modulo computations. For this dataset, layer-wise pre-training was counterproductive for fine-tuning and only two-layer methods could solve the task.

To obtain unrestricted representational power, we employed two-layer encoders, which can be regularized using an adaption of the contractive regularizer (Rifai et al., 2011c) and combined with one-layer encoders by using linear transformations and the powerful learning algorithm developed by Raiko et al. (2012) and Vatanen et al. (2013).

We empirically demonstrated the validity of our claim by showing that on a task of classifying handwritten digits, pre-training with two-layer encoders resulted both in better average performance under the same hyper-parameter prior and better absolute performance. For semi-supervised learning, when only a small number of training samples out of training samples are assumed to have annotated labels, we showed that pre-training indeed helps significantly. Furthermore, we were able to see that generalization performance could be improved by pre-training an MLP with a two-layer contractive encoding using the linear transformation, further confirming the validity of our claim.

Acknowledgment

This work was supported by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127.
Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, Special Issue on Learning Deep Architectures, early Access.

Bengio, Y., & Delalleau, O. (2011). On the expressive power of deep architectures. In *Algorithmic learning theory* (pp. 18–36). Springer.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. In *Proc. of advances in neural information processing systems* (pp. 153–160).

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Proc. of advances in neural information processing systems* (pp. 153–160). Cambridge, MA: MIT Press.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (Eds.), *Proc. of advances in neural information processing systems* (pp. 2546–2554).

Cho, K., Raiko, T., & Ilin, A. (2011). Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In *Proc. of the int. conf. on machine learning* (pp. 105–112).

Cireşan, D. C., Meier, U., & Schmidhuber, J. 2012. Multi-column deep neural networks for image classification. In *Proc. of the IEEE conf. on computer vision and pattern recognition* (pp. 3642–3649).

Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, June. arXiv:1406.2572 [cs.LG].

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

Erhan, D., Manzagol, P., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proc. of the int. conf. on artificial intelligence and statistics* (pp. 153–160).

Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on Information Theory*, *8*(1), 21–28.

Glorot, X., & Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the int. conf. on artificial intelligence and statistics* (pp. 249–256).

Glorot, X., & Bengio, Y. (2010b). Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the int. conf. on artificial intelligence and statistics, Vol. 9 of JMLR workshop and conf. proc. JMLR W&CP* (pp. 249–256).

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Le Roux, N., & Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, *20*(6), 1631–1649.

Minsky, M., & Seymour, P. (1969). *Perceptrons*. MIT press.

Ngiam, J., Chen, Z., Koh, P. W., & Ng, A. Y. (2011). Learning deep energy models. In *Proc. of the int. conf. on machine learning* (pp. 1105–1112).

Raiko, T., Valpola, H., & LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *Proc. of the int. conf. on artificial intelligence and statistics, Vol. 22 of JMLR workshop and conf. proc. JMLR W&CP, April* (pp. 924–932).

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Proc. of advances in neural information processing systems* (pp. 1137–1144). Cambridge, MA: MIT Press.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., & Glorot, X. (2011a). Higher order contractive auto-encoder. In D. Gunopulos, T. Hofmann, D. Malerba, et al. (Eds.), *Lecture Notes in Computer Science: Vol. 6912. Proc. of the European conf. on machine learning and knowledge discovery in databases* (pp. 645–660). Berlin, Heidelberg: Springer.

Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., & Dauphin, Y. et al. (2011b). Higher order contractive auto-encoder. In *Proc. of the European conf. on machine learning and knowledge discovery in databases* (pp. 645–660).

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011c). Contractive auto-encoders: explicit invariance during feature extraction. In *Proc. of the int. conf. on machine learning* (pp. 833–840).

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011d). Contractive auto-encoders: explicit invariance during feature extraction. In L. Getoor, & T. Scheffer (Eds.), *Proc. of the int. conf. on machine learning* (pp. 833–840). New York, NY, USA: ACM.

Salakhutdinov, R., & Hinton, G. (2009). Deep Boltzmann machines. In *Proc. of the int. conf. on artificial intelligence and statistics* (pp. 448–455).

Schulz, H., & Behnke, S. (2012). Learning two-layer contractive encodings. In *Proc. of the int. conf. on artificial neural networks* (pp. 620–628).

Spall, J. C. (1998). An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, *19*(4), 482–492.

Vatanen, T., Raiko, T., Valpola, H., & LeCun, Y. (2013). Pushing stochastic gradient towards second-order methods—backpropagation learning with transformations in nonlinearities, May. arXiv:1301.3476 [cs.LG].