In Proceedings of International Conference on Artificial Neural Networks (ICANN), pp. 620-628, September 2012.

Learning Two-Layer Contractive Encodings

Hannes Schulz and Sven Behnke

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik VI, Friedrich-Ebert-Allee 144, 53113 Bonn
{schulz,behnke}@ais.uni-bonn.de

Abstract. Unsupervised learning of feature hierarchies is often a good initialization for supervised training of deep architectures. In existing deep learning methods, these feature hierarchies are built layer by layer in a greedy fashion using auto-encoders or restricted Boltzmann machines. Both yield encoders, which compute linear projections 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 not restricted in the type of features it can learn. The proposed encoder can be regularized by an extension of previous work on contractive regularization. We demonstrate the advantages of two-layer encoders qualitatively, as well as on commonly used benchmark datasets.

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 [1, 2] and has spurred research on understanding and improving feature learning methods [e.g. 3–6]. 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. [3] 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. [6] showed that *stable* features of the training data are useful for supervised training on wide range of datasets. 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, Minsky and Seymour [7] showed that such a one-layer neural network 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 are of the XOR class (which we shall refer to as *non-linear* features), and that fine-tuning cannot repair this defect. We construct problems that cannot profit from pre-training and show that pre-training is even counter-productive in these cases.

The 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 [6] to two-layer auto-encoder pre-training. We show that this regularization can resolve the constructed cases and performs better than greedy pre-training on benchmark datasets.

The remainder of the paper is organized as follows. Sec. 2 discusses related work. In Sec. 3, we introduce technical details of auto-encoders and the pre-training protocol. Sec. 4 shows where greedy pre-training of auto-encoders fails and introduces our proposed solution. We demonstrate the benefits of our approach in the experiments of Sec. 5.

2 Related Work

The representational power of deep architectures has been thoroughly analyzed [8, 9]. Le Roux and Bengio [8] 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 which *features* can be represented, however, is not addressed. Bengio and Delalleau [9] analyzed the representational power of deep architectures and find they can represent some functions with exponentially less units than shallow architectures. However, the authors did not take into account the fundamental limits of their building blocks.

There is numerous evidence that performance of deep architectures can be improved when the greedy initialization procedure is relaxed. Salakhutdinov and Hinton [10] report advantages when performing simultaneous unsupervised optimization of all layer parameters of a stack of RBMs. The authors rely on a greedy initialization, however, which we demonstrate here might establish a bad starting point. Ngiam et al. [11] 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 et al. [12] obtained top results on a number of image classification data sets 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 Auto-Encoders

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

$$\mathbf{h} = f_{\text{enc}}(\mathbf{x}) = \mathbf{g}(W\mathbf{x}), \tag{1}$$

where $g_i(\mathbf{x}) = (1 + \exp(-x_i))^{-1}$ is a component-wise sigmoid non-linearity. 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 $W \in \mathbb{R}^{M \times N}$. The decoder is then given by

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

where we restrict ourselves to the symmetric case $W' = W^T$. Even though biases are commonly used, we omit them here for clarity. The main objective for auto-encoders is to determine W such that $\hat{\mathbf{x}}$ is similar to $\mathbf{x}$. For binary $\mathbf{x}$, this amounts to minimizing the the cross-entropy loss

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

Auto-encoders have gained popularity as a method for pre-training of deep neural networks [e.g. 1]. In deep neural networks, gradient information close to the input layer is "diluted" since it was passed through a series of randomly initialized layers. This effect increases the difficulties in learning good high-level representations [13]. Pre-training moves weights to an area where they relate to the input and therefore allow for cleaner gradient propagation.

Pre-training typically proceeds in a greedy fashion. Consider a deep network for a classification task, $\mathbf{y} = \mathbf{g}(V\mathbf{h}^{(L)})$, with $\mathbf{h}^{(l)} = \mathbf{g}(W^{(l)}\mathbf{h}^{(l-1)})$, where $\mathbf{y}$ is the output layer, $\mathbf{h}^{(l)}$, $l \in 1, \ldots, L$ are hidden layers, and $\mathbf{h}^{(0)} = \mathbf{x}$ is the input layer. Then, L auto-encoders are learned in succession, minimizing $\ell_\cdot(\mathbf{h}^{(l-1)}, W^{(l)})$ and keeping $W^{(l'<l)}$ fixed. It is frequently observed that following this protocol, successively higher-level representations of the inputs are attained [1]. This pre-training is greedy though; a joint optimization of all layers might yield superior results in theory. In practice, however, joint optimization suffers from the same gradient dilution problem as originally addressed by the deep-learning methodology and yields bad performance [14]. After pre-training, all parameters $(W^{(1)}, \ldots, W^{(L)}, V)$ are optimized according to the supervised classification objective. This final step is termed *fine-tuning*.

3.1 Contractive Auto-Encoder Regularization

To avoid overfitting the training data, or in the overcomplete case where $M > N$, it is common to regularize the auto-encoder learning. One possibility is to extend the training set by corrupting $\mathbf{x}$ with random noise and optimizing a denoising objective [15]. A more recent method is the contractive auto-encoder (CAE) [6].

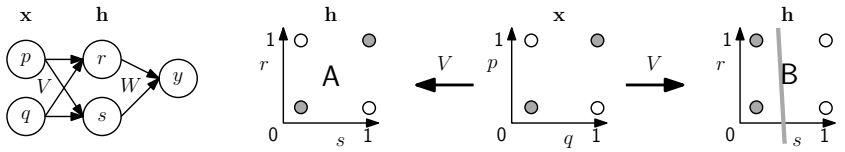

Fig. 1. Auto-encoder pre-training can be counter-productive. The simple network on the left should learn $y = p \veebar q$. Pre-training of V makes an uninformed choice between representations A and B (w.l.o.g.), but only B is linearly separable and helps fine-tuning.

The CAE minimizes the squared Frobenius norm of the Jacobian $J_{f_{\text{enc}}}$ of f_{enc} w. r. t. the input $\mathbf{x}$. The combined objective is given as

$$\ell_{CAE}(\mathbf{h}^{(l-1)}, W) = \ell_{\cdot}(\mathbf{h}^{(l-1)}, W) + \lambda \|J_{f_{\text{enc}}}\|^2, \tag{4}$$

where λ is the regularization strength. This regularizer yields very good results on a variety of datasets by identifying features which are invariant to small perturbations of the inputs. In the next section, we show that not all stable features can be computed by a CAE, severely restricting the types of invariances which can be learned.

4 Where Pre-Training of One-Layer Encoders Fails

Let us assume that we want to approximate the Boolean function $f(p, q) := p \veebar q$, 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 V. During the pre-training phase, two filters have to be learned, mapping the input vector again to a two-dimensional space $\mathbf{h} = (r, s)$. Without further information, this mapping might choose a representation which is not linearly separable (denoted "A" 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 $\mathbf{x}^{(i)} = (p, q, p \veebar q)$. The only stable feature of this dataset is $p \veebar q$, i.e. a two-layered denoising auto-encoder should be able to recover $x_3^{(i)}$ from the first two components and any of the p, q from the other variable and $p \veebar q$. If unfortunate greedy training of the first layer prevents the second layer from learning that p and q are XOR-related—as demonstrated in the previous section—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 experimentally verify these claims in Sec. 5.

Table 1. Hyper-parameter distribution used in our experiments.

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

4.1 Contractive Regularizer for Two-Layer Encoders

Considering the failure mode of one-layer encoders discussed above, an intuitive extension is to add a hidden layer $\mathbf{h}' \in \mathbb{R}^K$ to the encoder, such that $f_{\text{enc}}(\mathbf{x}) = \mathbf{h} = \mathbf{g}(W\mathbf{h}') = \mathbf{g}(W\mathbf{g}(V\mathbf{x}))$,with $\mathbf{x} \in \mathbb{R}^M$, $V \in \mathbb{R}^{K \times M}$, $W \in \mathbb{R}^{N \times K}$. Extending contractive regularization [6] to two-layer encoders yields

$$\|J_{f_{\text{enc}}}(x)\|_F^2 = \sum_n^N \sum_m^M \big(h_n(1-h_n)\big)^2 \left(\sum_k^K w_{nk} v_{km} h'_k(1-h'_k)\right)^2 . \tag{5}$$

In contrast to other pre-training regularizers, Eq. (5) does not directly constrain the first-layer features. Instead, first-layer features are chosen in a way to ensure stable *second*-layer representations. Thus, our encoder is able to recognize non-linear features as being stable in a dataset. While the time complexity of the forward and backward pass for a (mini-) batch of size B is $O(B(NK+KM))$, the complexity of the regularizer amounts to $O(BNKM)$ for both passes. In practice, we reduce the computation time to $O(NKM)$ by calculating the regularization gradient only for one randomly selected instance in a batch.

5 Experiments

Our experiments follow a common protocol. For a fixed architecture, we repeatedly sample all free hyper-parameters from the distributions detailed in Tab. 1. Random search is faster at finding optima than grid or manual search [16]. For a weight matrix $W \in \mathbb{R}^{N \times M}$, weights are initialized uniformly with $w_{ij} \sim \mathcal{U}\left(-\sqrt{6/N+M}, \sqrt{6/N+M}\right)$ as proposed by Glorot and Bengio [5]. 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 Sec. 4 to a more realistic task. A low density parity check (LDPC) code, also known as Gallager code [17], 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

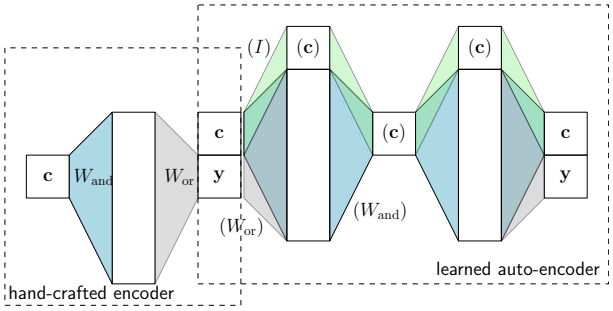

Fig. 2. Constructed example where greedy auto-encoders fail. The matrices $W_{\mathbf{and}}$ and $W_{\mathbf{or}}$ are hand-crafted to calculate $\sum_i c_i \mod 2$. Matrices and resulting representations in parenthesis have to be recovered, i. e. learned, by the auto-encoder to solve the reconstruction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

to both sender and receiver. A constraint over a set of variables X_c is met iff $0 \equiv (\sum_{x \in X_c} x) \mod 2$. 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 $\mathbf{c} \in \{0,1\}^N$ has N constraints C_n with three participating variables, each. Each variable participates in three random constraints. The problem can be solved perfectly by constructing a two-layer neural network with weights $W_{\mathbf{and}} \in \mathbb{R}^{4N \times N}$ such that

$$\bigwedge_{n \in \{1,\dots,N\}} \bigwedge_{i \in \{0,\dots,3\}} 1 = h^{(1)}_{4n+i} \quad \text{iff} \sum_{c \in C_n} c = 2i, \text{ and } W_{\mathbf{or}} \in \mathbb{R}^{N \times 4N} \text{ s. t.}$$
$$\bigwedge_{n \in \{1,\dots,N\}} 1 = y_n \quad \text{iff} \sum_{i \in \{0,\dots,3\}} h^{(1)}_{4n+i} > 0.$$

We now frame learning $W_{\mathbf{and}}$ and $W_{\mathbf{or}}$ as a task for the auto-encoder shown in Fig. 2. The auto-encoder reconstructs the code word $\mathbf{c}$ and the constraint violation vector $\mathbf{y}$ together:

$$(c_1, c_2, \dots, c_n, y_1, y_2, \dots, y_n) = \mathbf{x} = f_{\text{dec}}(f_{\text{enc}}(\mathbf{x})), \tag{6}$$
$$\text{where } f_{\text{enc}}(\mathbf{x}) = \mathbf{g}(W_{\mathbf{or}}\, \mathbf{g}(W_{\mathbf{and}}\, \mathbf{x})) \tag{7}$$
$$\text{and } f_{\text{dec}}(\mathbf{h}) = \mathbf{g}(W^T_{\mathbf{and}}\, \mathbf{g}(W^T_{\mathbf{or}}(\mathbf{h}))). \tag{8}$$

Since $c_i \sim \text{Binomial}(1, 0.5)$, $\mathbf{c}$ 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, the reconstruction loss $\ell_{\text{bin}}(\mathbf{x}, (W_{\mathbf{or}}, W_{\mathbf{and}}))$ is finetuned without regularization—again using early stopping on the validation set. An instance $\mathbf{x}$ is classified correctly if it could be reconstructed without errors, i. e. $\bigwedge_i (x_i = \lfloor \hat{x}_i + 0.5 \rfloor)$.

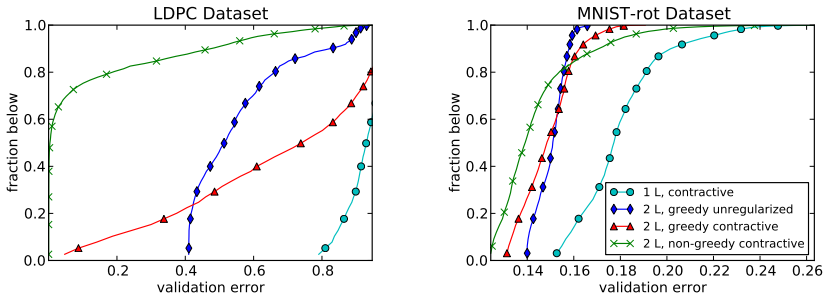

Fig. 3. Comparison of pre-training effects for fixed architecture (30×75×15 for LDPC, 784×1000×500 for MNIST-rot). Graphs show fraction of draws from hyper-parameter distribution which performs better than given validation error.

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
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 results are visualized in Fig. 3. 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 Tab. 2. Only conditions where both layers are trained *simultaneously* are able to solve the task. This demonstrates: Greedy auto-encoders are only able to learn "linear" features. Non-linear relations contained in the data cannot always be recovered by higher layers.

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 $N = 784$, $K = 1000$, $M = 500$ 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. 3) and finds better minima (Tab. 2). Additionally, we analyzed the reconstruction error of the best-performing models after pre-training. 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.

6 Conclusion

Common pre-training of deep architectures by RBM and AE simplifies one hard deep problem to many 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 used two-layer encoders, which can be regularized using an adaption of the contractive regularizer [6]. We demonstrated the superior performance of our approach on standard benchmark datasets, and compared to one-layer and greedy two-layer approaches.

References

[1] Y. Bengio et al. "Greedy layer-wise training of deep networks". In: *NIPS*. 2006, pp. 153–160.

[2] G. Hinton et al. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[3] D. Erhan et al. "The difficulty of training deep architectures and the effect of unsupervised pre-training". In: *AISTATS*. 2009, pp. 153–160.

[4] K. Cho et al. "Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines". In: *ICML*. 2011.

[5] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *AISTATS*. 2010.

[6] S. Rifai et al. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *ICML*. 2011.

[7] M. Minsky and P. Seymour. *Perceptrons*. MIT press, 1969.

[8] N. Le Roux and Y. Bengio. "Representational power of restricted boltzmann machines and deep belief networks". In: *Neural Computation* 20.6 (2008), pp. 1631–1649.

[9] Y. Bengio and O. Delalleau. "On the expressive power of deep architectures". In: *Algorithmic Learning Theory*. Springer. 2011, pp. 18–36.

[10] R. Salakhutdinov and G. Hinton. "Deep Boltzmann machines". In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics*. Vol. 5. 2. 2009, pp. 448–455.

[11] J. Ngiam et al. "Learning deep energy models". In: *ICML*. 2011, pp. 1105–1112.

[12] D. C. Cireşan et al. "Multi-column deep neural networks for image classification". In: *CVPR*. to appear. Providence, USA, 2012.

[13] D. Erhan et al. "Why does unsupervised pre-training help deep learning?" In: *Journal of Machine Learning Research* 11 (2010), pp. 625–660.

[14] G. Hinton and R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[15] P. Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *The Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[16] J. Bergstra and Y. Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[17] R. Gallager. "Low-density parity-check codes". In: *Information Theory, IRE Transactions on* 8.1 (1962), pp. 21–28.