Computer Science • 16 (4) 2015 http://dx.doi.org/10.7494/csci.2015.16.4.313

Karol Grzegorczyk
Marcin Kurdziel
Piotr Iwo Wójcik

# EFFECTS OF SPARSE INITIALIZATION IN DEEP BELIEF NETWORKS

**Abstract** *Deep neural networks are often trained in two phases: first, hidden layers are pretrained in an unsupervised manner, and then the network is fine-tuned with error backpropagation. Pretraining is often carried out using Deep Belief Networks (DBNs), with initial weights set to small random values. However, recent results established that well-designed initialization schemes, e.g., Sparse Initialization (SI), can greatly improve the performance of networks that do not use pretraining. An interesting question arising from these results is whether such initialization techniques wouldn't also improve pretrained networks. To shed light on this question, in this work we evaluate SI in DBNs that are used to pretrain discriminative networks. The motivation behind this research is our observation that SI has an impact on the features learned by a DBN during pretraining. Our results demonstrate that this improves network performance: when pretraining starts from sparsely initialized weight matrices, networks achieve lower classification errors after fine-tuning.*

**Keywords** Sparse Initialization, Deep Belief Networks, Noisy Rectified Linear Units

**Citation** Computer Science 16 (4) 2015: 313–327

## 1. Introduction

For many years, the primary method for training feed-forward neural networks was the error backpropagation algorithm [14]. Until recently, however, backpropagation was unsuccessful in training deep neural networks (i.e., neural networks with more than one hidden layer) to high levels of performance. This began to change in 2006 when Hinton & Salakhutdinov [7] demonstrated that deep neural networks can be trained in two phases: greedy layer-wise pretraining followed by fine-tuninig with error backpropagation. The pretraining in this ground-breaking result was unsupervised, i.e., did not employ class labels, and its goal was to capture the hierarchy of features extracted from the training data. Hinton and Salakhutdinov used stacked *Restricted Boltzmann Machines* (RBMs) [15] to pretrain their networks. Since then, pretrained deep neural networks demonstrated state-of-the-art performance in many machine learning tasks.

More recently, several works explored the possibility of training deep neural networks without resorting to layer-wise pretraining. An important lesson from these works is that training deep networks using only error backpropagation can lead to competitive performance, provided that the network is initialized with a well-designed random initialization scheme and a strong optimization method is used to backpropagate errors [18]. The most promising results along these lines were achieved using the *Sparse Initialization* (SI) technique proposed by Martens in [10].

An interesting question arising from recent successes with networks that do not use pretraining is whether initialization techniques developed therein wouldn't also improve pretrained networks. In particular, in this work we are interested in effects of Martens' SI on networks pretrained with RBMs. We, therefore, use SI to initialize stacked RBMs before pretraining and fine-tuning them in discriminative settings. We then compare the performance of networks trained in this manner with pretrained networks that use currently-advised wholly-random initialization. Our results demonstrate that SI improves performance also in pretrained networks, indicating that it might be a better initialization scheme for these networks than the currently used random initialization.

## 2. Background

Restricted Boltzmann Machine [15] is a generative model with units arranged in a bipartite graph. It assigns to every configuration $(\mathbf{v}, \mathbf{h})$ of visible and hidden units a probability of:

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{\mathbf{v}\mathbf{a}^{\mathrm{T}} + \mathbf{h}\mathbf{b}^{\mathrm{T}} + \mathbf{v}\mathbf{W}\mathbf{h}^{\mathrm{T}}}}{\sum\limits_{\mathbf{v}', \mathbf{h}'} e^{\mathbf{v}'\mathbf{a}^{\mathrm{T}} + \mathbf{h}'\mathbf{b}^{\mathrm{T}} + \mathbf{v}'\mathbf{W}\mathbf{h}'T}} \tag{1}$$

where $\mathbf{W}$ is a weight matrix for connections between visible and hidden units, $\mathbf{a}$ is a vector of visible unit biases and $\mathbf{b}$ is a vector of hidden unit biases. In the simplest case, visible and hidden units are binary. That is: $v_i, h_j \in \{0, 1\}$, $i = 1 \ldots n$,

$j = 1 \ldots m$, where $n$ is the number of visible units and $m$ is the number of hidden units. Then, the probability that a unit is on is typically given by the sigmoid nonlinearity: $p\left(h_j = 1|\mathbf{v}\right) = \left(1 + e^{-\left(\mathbf{v}\mathbf{W}_{\cdot j} + b_j\right)}\right)^{-1}$, $p\left(v_i = 1|\mathbf{h}\right) = \left(1 + e^{-\left(\mathbf{h}\mathbf{W}_{i\cdot}^{\mathrm{T}} + a_i\right)}\right)^{-1}$, where $\mathbf{W}_{i\cdot}$ and $\mathbf{W}_{\cdot j}$ are the i-th row and the j-th column of the weight matrix $\mathbf{W}$, respectively. Note, however, that certain other activation functions can also be used with RBMs, e.g., to model non-binary vectors [6].

During its training, RBM learns the probability distribution of the observed data. To this end, RBM training algorithms approximate the gradient of the log-likelihood of training vectors with respect to the weights and biases. One of the most commonly used approximations to this gradient is the *Contrastive Divergence* (CD) algorithm [5]. A training step in CD begins with taking a sample of visible and hidden units over the training data. The algorithm thus picks a random training example $\mathbf{v}^{(p)}$ and then takes a sample $\mathbf{h}^{(p)}$ of hidden units according to the activation probabilities $p\left(h_j^{(p)} = 1|\mathbf{v}^{(p)}\right)$. Next, CD takes an approximate sample $\left(\mathbf{v}^{(n)}, \mathbf{h}^{(n)}\right)$ from the RBM model by performing alternating Gibbs sampling of the visible and hidden units, starting the chain from the hidden configuration $\mathbf{h}^{(p)}$. The gradient is then approximated as:

$$\begin{aligned}
\frac{\partial \log p\left(\mathbf{v}^{(p)}\right)}{\partial \mathbf{W}} &= {\mathbf{v}^{(p)}}^{\mathrm{T}}\mathbf{h}^{(p)} - {\mathbf{v}^{(n)}}^{\mathrm{T}}\mathbf{h}^{(n)} \\
\frac{\partial \log p\left(\mathbf{v}^{(p)}\right)}{\partial \mathbf{a}} &= \mathbf{v}^{(p)} - \mathbf{v}^{(n)} \\
\frac{\partial \log p\left(\mathbf{v}^{(p)}\right)}{\partial \mathbf{b}} &= \mathbf{h}^{(p)} - \mathbf{h}^{(n)}
\end{aligned} \tag{2}$$

In its fastest variant CD performs only one Gibbs step – the so-called $\mathrm{CD}_1$ algorithm.

$\mathrm{CD}_1$ was used by Hinton & Salakhutdinov [7] to train *Deep Belief Networks* (DBNs), i.e., stacked RBMs where the first RBM layer models observed data and each subsequent RBM models outputs from the previous layer. This procedure was used to obtain initial weights for deep autoencoders and deep multilayer perceptron networks. Networks initialized in this manner were then fine-tuned with error backpropagation, ultimately achieving state-of-the-art performance on several dimensionality reduction and classification tasks.

The pretraining procedure described in [7] was further developed by Nair & Hinton [11] with the introduction of *Noisy Rectified Linear Units* (NReLUs), i.e., units with an activation function given by:

$$\mathrm{NReLU}\left(x\right) = \max\left\{0, x + N\left(0, \left(1 + e^{-x}\right)^{-1}\right)\right\} \tag{3}$$

Here, $N\left(m, s^2\right)$ denotes a random variable drawn from a Gaussian distribution with mean $m$ and variance $s^2$. NReLUs replace binary hidden units during layer-wise pretraining. Afterwards, when the network is fine-tuned with error backpropagation,

hidden layers employ a deterministic variant of the above activation function, i.e.:

$$\mathrm{ReLU}(x) = \max\{0, x\} \tag{4}$$

Nair & Hinton demonstrated that networks with NReLUs in hidden layers outperform networks with binary hidden layers on image classification tasks [11].

## 3. Related work

While layer-wise pretraining followed by fine-tuning with error backpropagation proved to be an effective way of training deep neural networks, certain recent works have focused on training such networks without the pretraining step. One lesson from these works is that in networks that do not employ pretraining, the choice of a random initialization scheme used for generating the initial weights has a significant impact on the outcome of the training. In [4], Glorot & Bengio show why standard random initialization is unsuited for error backpropagtion in deep networks, especially with sigmoid activations. Then, they propose *normalized initialization*, where an $n \times m$ weight matrix is drawn from a uniform distribution between $\pm\sqrt{6/(n+m)}$, and show that it can improve performance of deep networks with hyperbolic tangent activations. In [10], Martens proposed the *Hessian-free* optimization method for backpropagation networks. While this algorithm outperforms standard stochastic gradient descent, it too benefits from a well-designed random initialization scheme. In particular, the best results in [10] were obtained with there-proposed Sparse Initialization (SI) approach. SI initializes units with sparse, randomly generated weight vectors. To this end, a fixed number of elements are randomly chosen in each weight vector. These elements are initialized with random weights, usually drawn from a Gaussian distribution. All other elements in the weights matrix are set to zero. SI was designed to fulfill two goals: prevent the saturation of network units, and make the units initially as different from each other as possible. Further evidence for the importance of initialization in networks with no pretraining was given by Sutskever et al. in [18]. This work demonstrated that, under certain conditions, stochastic gradient descent can match the performance of Martens's Hessian-free optimizer. The conditions for these performance levels were the use of proper random initialization (which was SI in this case) and the use of momentum method [13] or *Nesterov Accelerated Gradient* [12] during training.

Works cited above study various forms of random initialization for networks with no pretraining. Comparatively less work has been published on initializing network layers before pretraining. Standard advice for RBM pretraining is to draw the initial weights from a Gaussian distribution with zero mean and a small standard deviation [6]. Also, with binary visible units, CD pretraining can be sped up by adjusting the visible biases to the mean input on the training set [6]. Nevertheless, Bergstra & Bengio [2] carried out experiments that included choosing the initial weights for RBM pretraining from several different random number distributions. The goal there was to compare a random search for network hyper-parameters with manually-assisted

hyper-parameter optimization. When pretraining and fine-tuning stacked RBMs, Bergstra & Bengio do not claim improvement over standard, manually-tuned hyper-parameters, and conclude that in these networks, only a small fraction of sensible hyper-parameters yield the best performance.

## 4. Sparse Initialization in Deep Belief Networks with Noisy Rectified Linear Units

When pretraining DBNs, dense random initialization with small weights typically gives good results. As we noted in the previous section, this kind of initialization is currently advised for DBNs. Yet, given the recent results on initialization in networks with no pretraining, one can ask whether pretraining couldn't also benefit from better initial weights. In particular, in this work we are interested in the effects of Martens's Sparse Initialization on networks pretrained with CD.

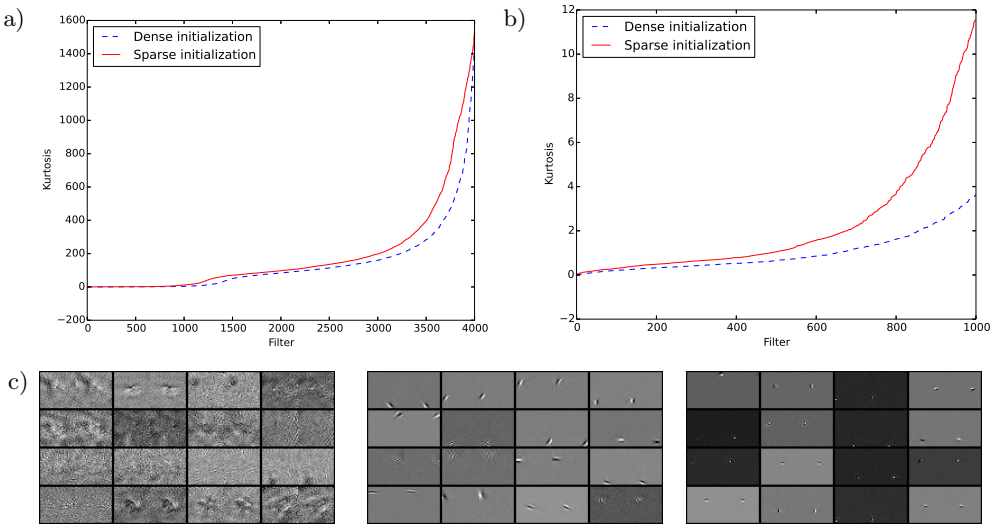


**Figure 1.** Kurtosis of filters (i.e., weight vectors of hidden units) learned by an RBM on the NORB dataset, when initial weights were either drawn from a Gaussian distribution with zero mean and small standard deviation (blue) or chosen with Sparse Initialization (red) (a). First 1000 filters from plot a. Filters on plots a and b are sorted according to their kurtoses (b). Sample filters with kurtosis less than 10 (left), between 50 and 100 (center) and greater than 200 (right). Note that kurtosis of an RBM filter corresponds to the kind of feature learned by the filter (c).

Our motivation to investigate SI in pretrained networks is that with SI hidden units are initially very different from each other and, we hope, they could learn a more diverse set of features than with classical dense initialization. Further motivation stems from the observation that RBMs with NReLUs in the hidden layer

are quite sensitive to initial weights. For example, we have repeated the pretraining experiments reported by Nair & Hinton [11] for the NORB dataset [9], using either dense initialization (with weights drawn from $N\left(0, 0.01^2\right)$ distribution) or Sparse Initialization, where each unit had 15 non-zero weights (also drawn from $N\left(0, 0.01^2\right)$). As noted by Nair & Hinton, on this dataset RBM learns Gabor-like edge detectors, point-like features and global filters. The kind of feature learned by a hidden unit is well reflected by kurtosis of its weight vector (Fig. 1c). By comparing kurtoses for the two initialization methods (plots on Figs 1a and 1b), we observe that with Sparse Initialization CD training gives less units with noisy global filters and more edge detectors or filters with point-like features.

While SI may have an impact on features learned by an RBM, it is not clear whether it may, in fact, improve the performance of pretrained neural networks. To shed light on this question, we carried out experiments on two standard machine learning datasets, where we compared the performance of pretrained and then fine-tuned DBNs that use either standard dense initialization or are initialized with SI. We report the results and details of these experiments in the subsequent sections.

## 5. Datasets and experiments

To evaluate SI in pretrained networks, we used two datasets commonly employed in machine learning research, namely MNIST [8] and Jittered-Cluttered NORB [9].

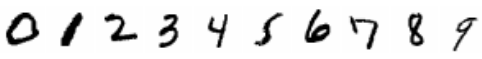

**Figure 2.** Example MNIST digits.

The challenge in the MNIST dataset is the recognition of handwritten digits (Fig. 2). It was built from a collection of digit images released by the U.S. National Institute of Standards and Technology. Images were preprocessed, resized, and put at the center of a $28 \times 28$ pixel window. Pixels in the MNIST dataset are represented by 255 grey-scale levels, but in our work pixel intensities are rescaled to the $\langle 0, 1 \rangle$ interval. The dataset comes in two standardized subsets, i.e., 60,000 training cases and 10,000 test cases. It is also common when working with this dataset to use 10,000 examples from the training set as a validation set and train the network with the remaining 50,000 examples.

The Jittered-Cluttered NORB dataset consists of 349,920 images that are split into 291,600 training cases and 58,320 test cases. Each image depicts one of 50 toys captured in stereo mode under variable lighting and viewpoints (Fig. 3). The images belong to six different classes, whose recognition is the task in this dataset. Note that training and test sets depict different toy instances. Furthermore, toy images are perturbed and presented on a complex background. These aspects of the dataset make the classification task harder. Following Nair & Hinton [11], we resized original images to $32 \times 64$ pixels, subtracted from each image its mean pixel intensity,

divided pixel intensities by the standard deviation of pixel intensities in the training set, and constructed a validation set consisting of 58,320 cases from the training set.

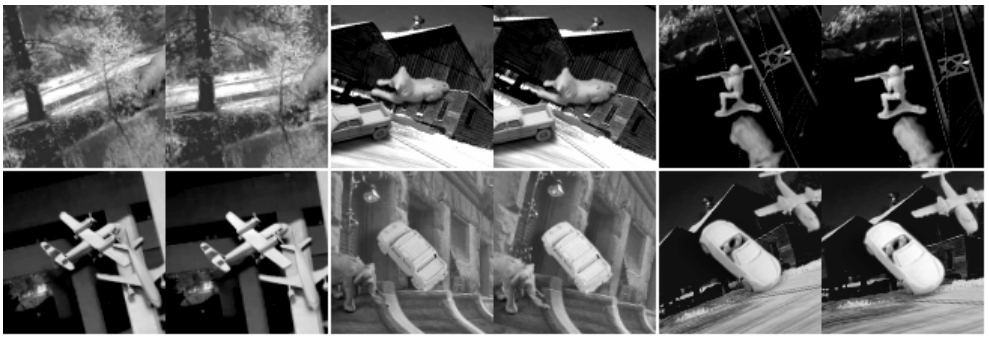

**Figure 3.** Example NORB images (one from each of the six classes).

For the MNIST experiments, we used one of the network architectures reported in [16]. It had 784 units in the input layer (which corresponds to the number of pixels in an input image) and two hidden layers, each with 1000 units. Visible units used the sigmoid activation function, while both hidden layers were made of NReLUs. When the network was fine-tuned for digit classification, the probability distribution for the ten different digits was represented by a 10-way *softmax* output layer [3]. The MNIST network was pretrained with 100 $CD_1$ epochs. Hyper-parameters for this phase were chosen following the recommendations in [6] and the results of evaluation on the validation set. To avoid overfitting, we used $\ell_2$ regularization term [6] in both layers. Momentum method [13] was used when updating weights and biases. The following other hyper-parameters were selected for pretraining: learning rate in the first layer $\lambda = 0.01$, learning rate in the second layer $\lambda = 0.001$, initial momentum $\mu = 0.5$, momentum after fifth epoch $\mu = 0.9$, and weight penalty $\ell_2 = 2 \cdot 10^{-5}$. In the fine-tuning phase the whole network was trained with error backpropagation. With experiments on the validation set, the following hyper-parameters were selected for this phase: learning rate in both hidden layers $\lambda = 0.03$, learning rate in the softmax layer $\lambda = 0.15$, and weight penalty in the softmax layer $\ell_2 = 5 \cdot 10^{-5}$. We also used momentum $\mu = 0.8$ in all layers. Pretraining and fine-tuning used stochastic gradient descent with mini-batches of size 128.

For the NORB dataset, we used the best performing network architecture reported by Nair & Hinton [11], i.e., 2048 Gaussian input units [6], two hidden layers with 4000 and 2000 NReLUs, respectively, and a 6-way softmax layer. Hyper-parameters for pretraining were either adopted from [11] or chosen with experiments on the validation set. That is, we pretrained each hidden layer using 300 $CD_1$ epochs, with learning rate $\lambda = 0.001$, weight penalty $\ell_2 = 2 \cdot 10^{-5}$, initial momentum $\mu = 0.5$, and momentum $\mu = 0.9$ after the fifth epoch. During fine-tuning, we used weight penalty $\ell_2 = 10^{-5}$ in the softmax layer, learning rates $\lambda = 0.01$ in hidden layers,

$\lambda = 0.1$ in the softmax layer, and momentum $\mu = 0.8$ in all layers. Both pretraining and fine-tuning were carried out using stochastic gradient descent with 128-element mini-batches.

When experimenting with SI in DBNs, we used a Gaussian distribution with zero mean for non-zero weights. This choice follows the SI scheme used by Sutskever et al. [18]. The biases were set to zero. In addition, we performed experiments on the validation sets to choose the standard deviation of the Gaussian distribution and the number of non-zero weights per hidden unit. For each dataset, we then selected three variants of SI that performed best on the validation set. These variants were evaluated on complete datasets, i.e., with pretraining and fine-tuning on the whole training set and evaluation on the test set. In addition to these main experiments, we also conducted tests with SI in the softmax layer. The tests were carried out with either SI or dense initialization in the hidden layers. In these experiments specific configurations of SI were also chosen following results from the validation sets. For the reference dense initialization we used a Gaussian distribution with zero mean and a small standard deviation of $\sigma = 0.01$.

Training deep neural networks requires significant computational resources. To speed up the training, we implemented the software used to carry out the reported experiments for Graphical Processing Units (GPUs). The implementation was done with the NVIDIA CUDA platform[1]. For most of the implementation we relied on the highly-optimized linear algebra kernels provided by the NVIDIA cuBLAS library. In particular, cuBLAS kernels were used for matrix-matrix and matrix-vector operations needed in CD and error backpropagation. The cuBLAS library uses column-major order; therefore, we adopted this order for all matrices in our code. Examples in mini-batch matrices are stored in rows. In weight matrices, weights vectors for the hidden units are stored column-wise. We also adopted a convention that each mini-batch matrix has an additional column filled with 1.0. This allows us to store biases for visible and hidden units in the last row and the last column of the weight matrix, respectively. Several element-wise matrix operations needed during training are not available in CUDA libraries, so we implemented them with our own kernels. In these kernels we adopted a domain decomposition in which each column is processed by one work-group. The size of the work-group was set to 128, which is equal to the size of our training mini-batches. To facilitate coalesced memory transactions and, thus, improve the performance of cuBLAS and our own kernels, we allocate all GPU matrices with proper padding added to each column. The padding ensures that matrix columns begin at 512-byte aligned addresses. The memory allocated to the padding is left unused. CD training requires the generation of large number of random numbers on the GPU. We employ device API of the NVIDIA cuRAND library to efficiently generate these numbers. The functionality of our GPU implementation is wrapped with Python bindings. The host-side data processing is carried out with the NumPy

[1] `http://www.nvidia.com/cuda`

library. We conducted the experiments reported in this work on NVIDIA Tesla M2090 cards, using CUDA platform v5.5, Python 2.7.5 and NumPy 1.8.1.

## 6. Results

Plots on Figures 4 and 5 report classification errors on the test sets during network fine-tuning. A summary of these results is given in Tables 1 and 1, where we report performance with the early stopping criterion as well as the best performance during fine-tuning. For the early stopping criterion, we use the epoch number in which the network performed best on the validation set.

**Table 1**

Classification errors on the test sets with different initialization schemes in the hidden layers. Results are reported for (a) MNIST and (b) NORB datasets. Early stopping is an error in the epoch with best performance on the validation set. Minimal error is the lowest error during fine-tuning.

a)

| **Weight initialization** | **Classification error [%]** | |
|---|---|---|
| | **Early stopping** | **Minimal** |
| Dense initialization | 1.15 | 1.12 |
| SI $n = 5$, $\sigma = 0.2$ | 1.07 | 1.06 |
| SI $n = 10$, $\sigma = 0.01$ | 1.04 | 1.02 |
| SI $n = 15$, $\sigma = 0.01$ | 1.08 | 1.04 |

b)

| **Weight initialization** | **Classification error [%]** | |
|---|---|---|
| | **Early stopping** | **Minimal** |
| Dense initialization | 14.75 | 14.68 |
| SI $n = 10$, $\sigma = 0.1$ | 14.38 | 14.33 |
| SI $n = 10$, $\sigma = 0.3$ | 14.39 | 14.37 |
| SI $n = 15$, $\sigma = 0.1$ | 14.37 | 14.32 |

On both datasets, networks that were initialized with SI before pretraining performed better than the networks with dense initialization. While the difference is notable on the more difficult NORB dataset, on the MNIST dataset Sparse Initialization also improved the already non-trivial performance. On the NORB dataset we also observe slightly faster backpropagation training when network is pretrained with SI. Different choices for the number of non-zero weights and their standard deviation led to slightly different performance, but all configurations of SI that we selected on the validation sets performed better than dense initialization. In addition to the reported results, we also carried out confirmatory experiments with eight different seed values for randomness in SI. These experiments demonstrated that differences in performance due to different random instances of SI are small and do not change the conclusions of this section – we observed a standard deviation of error values slightly below 0.13% on the NORB set and slightly below 0.04% on the MNIST set.

Improvement in network performance that we observe with SI comes from better pretraining of the hidden layers. In particular, results reported in Figures 6 and 7, and summarized in Table 2, show that SI in the classification layer does not lead to further performance gain.

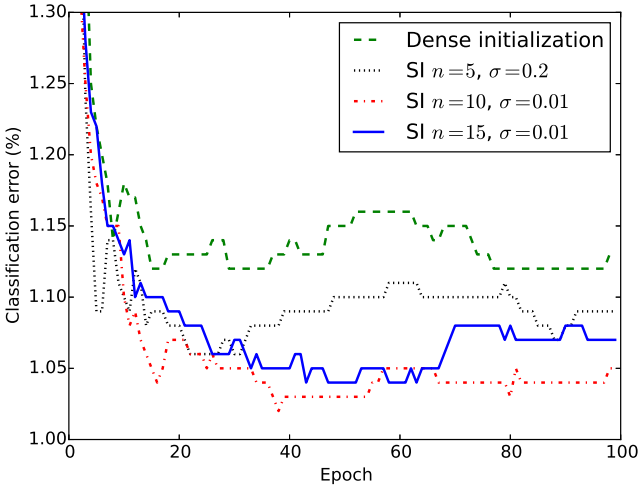


**Figure 4.** Classification errors on the MNIST test set with different initialization schemes in the hidden layers.

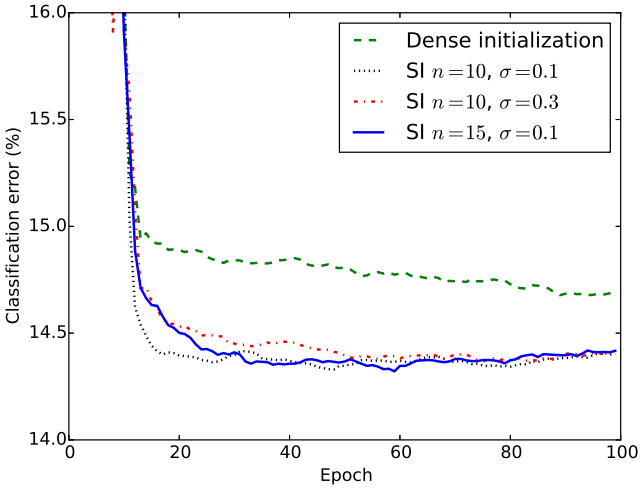


**Figure 5.** Classification errors on the NORB test set with different initialization schemes in the hidden layers.

**Table 2**

Classification errors on test sets with SI in the softmax layer. Results are reported for (a) MNIST and (b) NORB datasets. Early stopping is an error in the epoch with best performance on the validation set. Minimal error is the lowest error during fine-tuning.

a)

| Weight initialization | | Classification error [%] | |
|---|---|---|---|
| hidden layers | softmax layer | early stopping | minimal |
| Dense initialization | SI $n = 5$, $\sigma = 0.01$ | 1.14 | 1.12 |
| Dense initialization | SI $n = 10$, $\sigma = 0.01$ | 1.11 | 1.11 |
| Dense initialization | SI $n = 15$, $\sigma = 0.01$ | 1.12 | 1.12 |
| SI $n = 10$, $\sigma = 0.01$ | SI $n = 15$, $\sigma = 0.01$ | 1.07 | 1.06 |

b)

| Weight initialization | | Classification error [%] | |
|---|---|---|---|
| hidden layers | softmax layer | early stopping | minimal |
| Dense initialization | SI $n = 5$, $\sigma = 0.3$ | 14.71 | 14.68 |
| Dense initialization | SI $n = 10$, $\sigma = 0.01$ | 14.74 | 14.70 |
| Dense initialization | SI $n = 10$, $\sigma = 0.1$ | 14.63 | 14.61 |
| SI $n = 10$, $\sigma = 0.3$ | SI $n = 10$, $\sigma = 0.01$ | 14.34 | 14.32 |

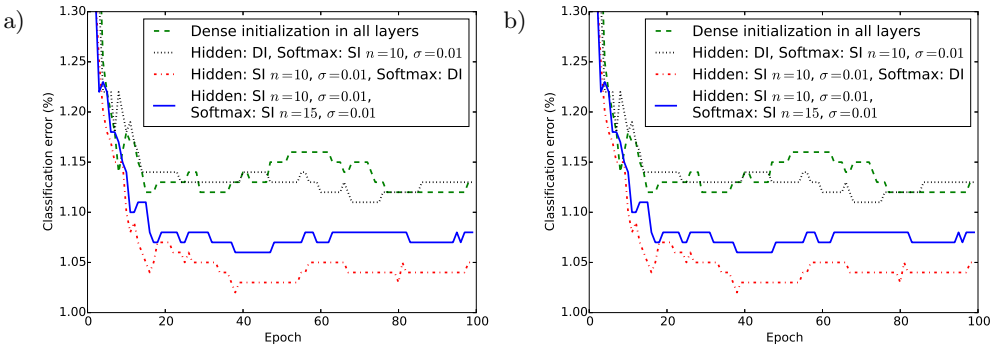


**Figure 6.** Classification errors on the MNIST test set with (a) SI in the softmax layer and (b) SI in the softmax and hidden layers. Plots for dense initialization in the softmax layer are included for comparison.

Networks that employed SI only in the classification layer performed similarly to the reference networks that used dense initialization in all layers (Figures 6a and 7a). On the other hand, when SI was used in all layers, performance was similar to networks that used SI only in the hidden layers (Figures 6b, 7b). Although plot on Figure 7a reports a decrease in classification error on the NORB dataset with one variant of SI in the softmax layer, this effect disappears when hidden layers are also initialized with SI (Figure 7b).

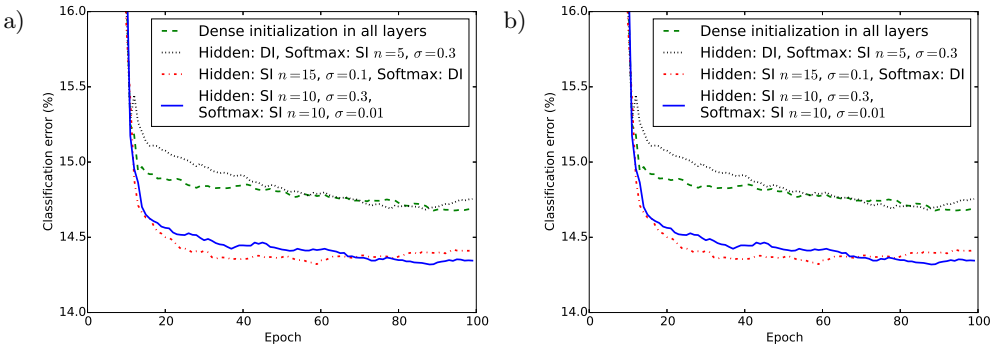


**Figure 7.** Classification errors on the NORB test set with (a) SI in the softmax layer and (b) SI in the softmax and hidden layers. Plots for dense initialization in the softmax layer are included for comparison.

## 7. Conclusions

Several works cited in section 3 report results that narrow (or even close) the gap between the performance of deep neural networks that do not use pretraining and networks studied by Hinton & Salakhutdinov [7]. Nevertheless, pretraining still serves a useful purpose in deep networks. For example, as reported by Martens [10], Hessian-free optimizer solves the problem of under-fitting in deep auto-encoders – and, in effect, surpasses performance levels reported in [7] – but pretraining improves generalization error obtained with this optimizer. A similar theme can be observed with the *dropout* regularization technique proposed in [17]. Therein, Hinton et al. report results for the MNIST dataset, where a network with dropout performed better than the pretrained network reported in [7]. However, an even lower classification error was obtained when dropout was used alongside pretraining [17]. Recommendations given in [1] also indicate that pretraining typically helps. It is therefore possible that certain techniques developed recently to improve performance of networks with no pretraining may lead to even better performance with pretrained networks. Our work goes in this direction. We studied the effects of Martens's Sparse Initialization [10] in DBNs employed to pretrain discriminative networks. This initialization scheme was originally developed to train deep networks starting from random weights (i.e., without pretraining) and demonstrated significant benefits in this application [18]. On the other hand, in networks with pretraining, the standard approach was to simply draw the initial weights from a Gaussian distribution with a small standard deviation [6]. Our results, however, demonstrate that SI may be useful also in pretrained networks. In particular, we demonstrate that DBNs with NReLUs in hidden layers benefit from SI, especially on difficult classification tasks.

Results in Sutskever et al. [18] show that, apart from initialization, momentum also plays an important role in deep networks with no pretraining. It is important

to note, however, that these results are mainly concerned with solving under-fitting problems in deep auto-encoders. Nevertheless, a possible direction of further research could be the evaluation of fine-tuning of DBNs with large momentum. While the usual approach here is to use a small learning rate and conservative momentum coefficient (e.g., $\mu = 0.8$) [7], once error on the training set becomes low, using large momentum could possibly improve training and the final classification performance.

## Acknowledgments

This research is supported by the Polish National Center of Science grant no. DEC-2013/09/B/ST6/01549 "Interactive Visual Text Analytics (IVTA): Development of novel, user-driven text mining and visualization methods for large text corpora exploration".

Special thanks are due to (partial) financial supported by the Polish Ministry of Science and Higher Education under AGH University of Science and Technology grant 11.11.230.124 (statutory project).

This research was carried out with the support of the "HPC Infrastructure for Grand Challenges of Science and Engineering" Project, co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme.

This research was supported, in part, by PL-Grid Infrastructure.

## References

[1] Bengio Y.: Practical Recommendations for Gradient-Based Training of Deep Architectures. In: G. Montavon, G.B. Orr, K.R. Müller, eds, *Neural Networks: Tricks of the Trade*, *Lecture Notes in Computer Science*, vol. 7700, pp. 437–478. Springer, Berlin–Heidelberg, 2012.

[2] Bergstra J., Bengio Y.: Random Search for Hyper-parameter Optimization. *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[3] Bridle J.S.: Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In: F. Soulié, J. Hérault, eds, *Neurocomputing*, *NATO ASI Series*, vol. 68, pp. 227–236. Springer, Berlin–Heidelberg, 1990.

[4] Glorot X., Bengio Y.: Understanding the difficulty of training deep feedforward neural networks. In: Y.W. Teh, M. Titterington, eds, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, vol. 9, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

[5] Hinton G.E.: Training products of experts by minimizing contrastive divergence. *Neural Computation*, vol. 14(8), pp. 1771–1800, 2002.

[6] Hinton G.E.: A Practical Guide to Training Restricted Boltzmann Machines. In: G. Montavon, G.B. Orr, K.R. Müller, eds, *Neural Networks: Tricks of the Trade*, *Lecture Notes in Computer Science*, vol. 7700, pp. 599–619. Springer, Berlin–Heidelberg, 2012.

[7] Hinton G.E., Salakhutdinov R.R.: Reducing the dimensionality of data with neural networks. *Science*, vol. 313(5786), pp. 504–507, 2006.

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

[9] LeCun Y., Huang F.J., Bottou L.: Learning methods for generic object recognition with invariance to pose and lighting. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR04)*, vol. 2, pp. II–97. IEEE, 2004.

[10] Martens J.: Deep learning via Hessian-free optimization. In: J. Fürnkranz, T. Joachims, eds, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 735–742. Omnipress, 2010.

[11] Nair V., Hinton G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: J. Fürnkranz, T. Joachims, eds, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814. Omnipress, 2010.

[12] Nesterov Y.: A method of solving a convex programming problem with convergence rate O (1/k2). *Soviet Mathematics Doklady*, vol. 27(2), pp. 372–376, 1983.

[13] Polyak B.T.: Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, vol. 4(5), pp. 1–17, 1964.

[14] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning representations by back-propagating errors. *Nature*, vol. 323(6088), pp. 533–536, 1986.

[15] Smolensky P.: Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: D.E. Rumelhart, J.L. McClelland, CORPORATE PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 194–281. MIT Press, 1986.

[16] Srivastava N.: *Improving neural networks with dropout.* Master's thesis, University of Toronto, 2013.

[17] Srivastava N., Hinton G.E., Krizhevsky A., Sutskever I., Salakhutdinov R.: Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, vol. 15(1), pp. 1929–1958, 2014.

[18] Sutskever I., Martens J., Dahl G., Hinton G.E.: On the importance of initialization and momentum in deep learning. In: S. Dasgupta, D. Mcallester, eds, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, vol. 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings, 2013.

## Affiliations

**Karol Grzegorczyk**
AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, Krakow, Poland, kgr@agh.edu.pl

**Marcin Kurdziel**
AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, Krakow, Poland,
kurdziel@agh.edu.pl

**Piotr Iwo Wójcik**
AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, Krakow, Poland, pwojcik@agh.edu.pl

**Received**: 16.10.2014
**Revised**: 18.12.2014
**Accepted**: 20.12.2014