Learning Understandable Neural Networks With Nonnegative Weight Constraints

Jan Chorowski, *Student Member, IEEE*, and Jacek M. Zurada, *Fellow, IEEE*

***Abstract*— People can understand complex structures if they relate to more isolated yet understandable concepts. Despite this fact, popular pattern recognition tools, such as decision tree or production rule learners, produce only flat models which do not build intermediate data representations. On the other hand, neural networks typically learn hierarchical but opaque models. We show how constraining neurons' weights to be nonnegative improves the interpretability of a network's operation. We analyze the proposed method on large data sets: the MNIST digit recognition data and the Reuters text categorization data. The patterns learned by traditional and constrained network are contrasted to those learned with principal component analysis and nonnegative matrix factorization.**

***Index Terms*— Multilayer perceptron, pattern analysis, supervised learning, white-box models.**

I. Introduction

HUMANS analyze complex relations by decomposing them hierarchically into isolated and understandable concepts. In contrast with this intuitive approach, computational tools for learning from data do not follow such hierarchies. Popular tools that create understandable models, such as decision tree or rule set inductors [1]–[3], produce only flat data descriptions with no hierarchy of concepts, while tools that build hierarchical models, such as artificial neural networks [4], do so in very convoluted ways that are inherently hard to understand [5]–[7]. We make an attempt at reconciling the requirements of hierarchical organization and interpretability by imposing positive-only weights which facilitate training of understandable neural networks. It has been suggested, that nonnegativity constraints may be a mechanism responsible for emergence of parts-based representations in the brain [8]. Here, we demonstrate this principle in a discriminative setting without any feature extraction step, usually accomplished using an unsupervised method, such as principal component analysis (PCA), independent component analysis [9], [10], or nonnegative matrix factorization) (NMF) [8], [11]. We find the results significant for two reasons. First, it is a step toward solving a long-standing open problem of understanding neural networks' processing. Second, it may shed light onto solutions of problems for which neural networks give specific good results, but convincing domain theory does not exist, such as protein secondary structure prediction [12].

Multilayer feed-forward neural networks build hierarchical models of data [4]. Nevertheless, two problems prevent them from being used to build understandable models. First, training of multilayer networks is a computationally demanding task. Second, it is usually difficult to interpret what the network has learned in each layer [5], [7]. The first problem has recently been addressed by the introduction of deep learning network initialization and training methods [13], [14]. Inspired by NMF, we address the second problem by trying, to constrain the network's weights to be nonnegative. This eliminates cancellations of incoming neuron signals inside the network and allows for easier interpretation. Hidden (input) neurons are active when their inputs correlate strongly with their weights. The bias controls the threshold of this correlation. Classification (output) layer neurons combine the hidden layer activation values in direct proportion to their weights, and the neuron with the highest sum determines the class of the input.

The problem of understanding neural networks or transforming them into models which show similar accuracy, but are easier to comprehend has long been studied. Pruning methods were designed to simplify networks by removing spurious units and connections inside a network [15]. OBD [16] and OBS [17] prune a trained network using second-order derivative information to estimate the impact of removing a connection. An algorithm that removes hidden nodes and adjusts remaining weights by solving a system of equations is presented in [18]. Other pruning methods expand the loss criterion minimized during network training with terms that promote the reduction of the number of connections or with terms that enforce other network simplifying constraints. Weight decay is the mechanism traditionally used to reduce the magnitude of network weights by penalizing the sum of their squares. It has been extensively studied both in the context of network understanding and network's generalization ability [19]–[21]. Enhanced sparsity of weights can be obtained by penalizing the sum of weights' absolute values instead of the sum of their squares [22]. This mechanism is similar to the elastic net feature selection technique used in linear regression [23]. Often particular values of network weights are required. Soft weight sharing [24] aims at clustering weight values. A polynomial penalty is used in [6] to constrain the

Manuscript received May 12, 2013; revised February 4, 2014; accepted February 22, 2014. Date of publication April 4, 2014; date of current version December 16, 2014.

J. Chorowski is with the Department of Mathematics and Computer Science, University of Wroclaw, Wroclaw 50-137, Poland, and also with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292 USA (e-mail: jan.chorowski@ii.uni.wroc.pl).

J. M. Zurada is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292 USA, and also with the Information Technology Institute, University of Social Sciences, Łódz 90-113, Poland (e-mail: jacek.zurada@louisville.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2014.2310059

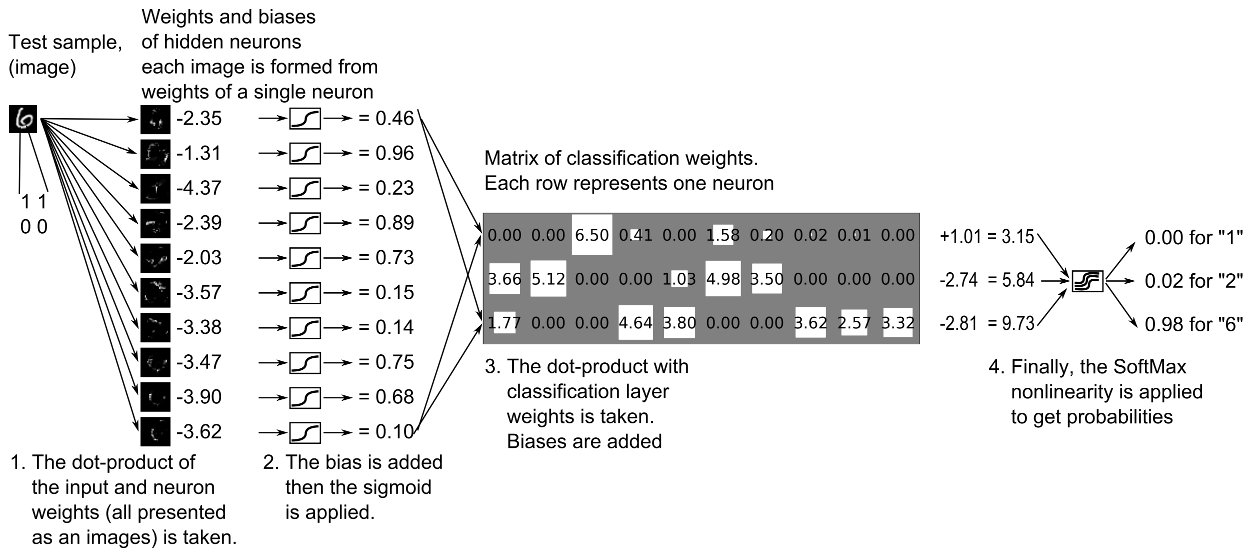

Test sample, (image)
Weights and biases of hidden neurons each image is formed from weights of a single neuron
1 1
0 0
-2.35
-1.31
-4.37
-2.39
-2.03
-3.57
-3.38
-3.47
-3.90
-3.62
= 0.46
= 0.96
= 0.23
= 0.89
= 0.73
= 0.15
= 0.14
= 0.75
= 0.68
= 0.10
Matrix of classification weights. Each row represents one neuron
0.00 0.00 6.50 0.41 0.00 1.58 0.20 0.02 0.01 0.00
3.66 5.12 0.00 0.00 1.03 4.98 3.50 0.00 0.00 0.00
1.77 0.00 0.00 4.64 3.80 0.00 0.00 3.62 2.57 3.32
+1.01 = 3.15
-2.74 = 5.84
-2.81 = 9.73
0.00 for "1"
0.02 for "2"
0.98 for "6"
1. The dot-product of the input and neuron weights (all presented as an images) is taken.
2. The bias is added then the sigmoid is applied.
3. The dot-product with classification layer weights is taken. Biases are added
4. Finally, the SoftMax nonlinearity is applied to get probabilities

Fig. 1. Signal propagation through the proposed network architecture. Input consists of 784 values that correspond to pixels of a 28×28 pixel images. Therefore, both the inputs and the 784 weights of every hidden neuron are presented as images.

weights to be zero or ± 1. Hyperbolic tangent nonlinearity has been applied to weights for the same purpose in [25]. Use of those techniques facilitates the understanding of the network because the analysis of interactions between signals incoming to a neuron is greatly simplified if the weights amplifying those signals are similar for all inputs. Rule extraction methods [5], [26], [27] can be used when a network has to be transformed into logical rules. Our work mostly resembles the weight penalty approach for neural network clarification because it consists of imposing nonnegativity constraints on weights of the network. However, while network pruning methods reduce the complexity of the network, they do not guarantee that hidden units of pruned network will have any identifiable meaning. Our approach differs in this respect since it is meant to produce networks, in which hidden nodes represent identifiable concepts.

Recently, there has been a resurgence of interest in automatic derivation of feature hierarchies from unlabeled data. The pioneering work trained restricted Boltzmann machines one layer at a time and stacked them to form a deep autoencoding network [13]. This greedy, one-layer at a time approach has been proved to be very general and applicable to other neural network architectures as well [14]. Several approaches of autoencoding network regularization were tested, including noise injection [28], sparsity [29], topography [30], and nonnegativeness [31]. This contribution applies some of these regularization techniques in a purely discriminative training. Therefore, two goals are attained: we confirm that these regularizations are useful in classical neural network training and we show that better discriminative feature extraction is possible.

II. Network Architecture

In this contribution, we study neural networks designed for classification and trained in a discriminative manner. We assume that the input data has nonnegative values. This condition is often satisfied in practice. For example, text documents in bag-of-words format or pixel intensities in images are naturally nonnegative. Categorical data encoded a using the 1-hot or thermometer-scale encoding is also nonnegative and can be used. The desired output for each sample must be a unique class label.

A. Notation and Network Details

We use networks with two layers. Each layer is determined by a matrix of weights and a vector of bias values, denoted by W_H and B_H for the hidden layer and by W_C and B_C for the classification (output) layer. For an input vector X the signal is propagated through the network according to the following relations. The hidden layer activations are $L_H(X) = \sigma(\lambda(W_H \cdot X + B_H))$, where $\sigma(x)$ denotes an element-wise application of the logistic sigmoid, $\sigma(x) = 1/(1+\exp(-x))$ and λ is a parameter denoting the neuron's gain. The classification layer activations are $L_C(X) = \mathrm{SoftMax}(W_C \cdot L_H(X) + B_C)$, where the $\mathrm{SoftMax}(V)$ function transforms a vector V into a vector of values according to $\mathrm{SoftMax}(V)_i = \exp(V_i)/(\sum_{j=1}^{n} \exp(V_j))$. The network assigns new data to the class represented by the output neuron with the highest activation value. The classification of an image of the digit 6 is presented in Fig. 1.

The use of the softmax activation function is important for two reasons. First, it is the generalization of the cross-entropy error function to multiple classes and is more suited for a classification task. It can be derived from an assumption that the class labels are discrete and mutually exclusive. Then, the outputs of the network using the softmax transfer function represent the *a posteriori* class probabilities for a given sample [32]. Second, the softmax function allows a degree of ambiguity for its inputs. A constant can be added to all inputs of a softmax function and the output will not be changed. We exploit this property in proving that the networks with nonnegative weights can learn any given labeling of data points.

It may seem that constraining the sign of weights of a neural network renders it too limited to be of any practical use. For instance, a neuron with positive weights and a logistic sigmoid activation function cannot logically invert its input. The proof given in the Appendix shows by construction that a three layer network using the logistic sigmoid activation for hidden layers with the softmax activation function used for the output layer can shatter any given set of points. The key insight is that the network's decision is based on the maximally active output neuron. While we cannot lower the output of a neuron associated with a wrong class by setting its weight to be negative, we can instead increase the outputs of all other classes. Due to the ambiguity present in the softmax function, subtracting a number from a neuron's activation is mathematically equivalent to adding the same number to all other activations. During learning this process tries to produce large weight values. We counteract it by adding weight regularization terms, which penalize their large values. These two counterbalancing processes cause the output layer to become sparse, while hidden neurons learn concepts which relate to parts of characteristics that define an output class.

B. Training Criterion

The network is trained by minimizing a loss function with respect to the weights and biases. The loss function used to train the network contains log-likelihood and regularization terms. The output L_C of the softmax transfer function sums to 1 and can be treated as the *a posteriori* probabilities of class labels given the input. We follow this interpretation and train the network by minimizing the negative log-likelihood of observing the given data set. Furthermore, the weights are regularized by minimizing their absolute values (ℓ_1 norm) and their squares (ℓ_2 norm) [22], [23], thus we employ a penalty-based weight pruning mechanism. The combined action of the ℓ_1 and ℓ_2 penalties both selects important connections and limits their magnitude. Sparse activations of the hidden layer can additionally be enforced by minimizing the sum of the hidden neuron activations. This further enhances readability—proper classification of a sample must depend on only a few hidden neurons becoming active. The complete optimization target is

$$\begin{aligned}\text{Loss} = &-\frac{1}{n}\sum_{S=1}^{n}\log\left[\left(L_C\left(X^S\right)\right)_{Y^S}\right]\\ &+\sum_{i,j}\left(p_{H1}|W_{H\,i,j}| + p_{H2}W_{H\,i,j}^2\right)\\ &+\sum_{j,k}\left(p_{C1}|W_{C\,j,k}| + p_{C2}W_{C\,j,k}^2\right)\\ &+\frac{p_S}{n}\sum_{S=1}^{n}\sum_{j}L_H\left(X^S\right)_j \end{aligned} \tag{1}$$

where n is the number of samples, $(X^S, Y^S) \in (\mathbb{R}^m \times \mathbb{N})$ are individual data samples and p_{H1}, p_{H2}, p_{C1}, p_{C2}, p_S are regularization constants. The first term of the optimization target is the cross-entropy error. Since for each sample Y^S is a natural number denoting the class label, $(L_C(X^S))_{Y^S}$ is network's output for the sample S of the neuron responsible for the proper class, i.e., Y^S. The next two terms regularize the weights by penalizing their magnitude, while the last term promotes the sparsity of hidden layer activations. We constrain the weight matrices W_H and W_C to contain only nonnegative elements. The bias values are unconstrained and often negative.

```
{The parameters are:
$BS$ – the batch size
$B$, $C$ – learning rate annealing parameters}
{$\theta$ is the vector of concatenated weights and biases.
Initialize $\theta$ with 0 for biases and small exponentially distributed random numbers for weights.}

$\theta \leftarrow \theta_0$ {Initial weight and biases}
$t \leftarrow 0$ {Iteration counter}
for prescribed number of epochs do
   permute training samples
   for all $b$ – consecutive batches of $BS$ train samples do
      $t \leftarrow t + 1$
      loss ← loss on samples in the batch $b$ from eq. (1)
      $\nabla\theta \leftarrow$ gradient of the loss on samples from $b$
      $\alpha \leftarrow \frac{1}{B+t\cdot C}$ {Anneal the learning rate}
      {Make a step along the gradient}
      $\theta \leftarrow \theta - \alpha\nabla\theta$
      {Project $\theta$ onto the feasible set}
      for all weight indexes $wi : \theta_{wi} < 0$ do
         $\theta_{wi} \leftarrow 0$
      end for
   end for
end for
```

Fig. 2. Stochastic projected gradient descent algorithm used to train networks with nonnegative weights.

C. Training Algorithm

Neural networks are typically trained by gradient descent on the loss function [4]. To train networks with nonnegative weights a similar approach can be used, however, the algorithm must ensure that the weights remain nonnegative. One solution is the projected gradient descent algorithm, which applies a step along the gradient and finds its constraint-satisfying projection [33]. Formally, let $\Omega \subseteq \mathbb{R}^k$ be the set of feasible solutions. In our case it is the set of k real numbers consisting of the weights and biases of the network, such that the weights are nonnegative. The projection $P : \mathbb{R}^k \to \Omega$ is defined as [33]

$$P(x) = \underset{z\in\Omega}{\arg\min}\ ||x - z||. \tag{2}$$

In the case of nonnegativity constraints the projection simply sets negative entries to zero. The full algorithm used to train the network is given in Fig. 2. It first initializes weights to small nonnegative random numbers (we have sampled the exponential distribution with mean 0.01). In each epoch training samples are shuffled and divided into batches of size BS (in our experiments BS = 10). On each batch the loss is computed using (1) (we use the batch size for n) and the derivative of the loss is computed using the back-propagation

```
{The parameters are:
$BS$ – the batch size}
{$\theta$ is the vector of concatenated weights and biases.
Initialize $\theta$ with 0 for biases and small exponentially distributed random numbers for weights.}

$\theta \leftarrow \theta_0$ {Initial weight and biases}
for prescribed number of epochs do
    for all $b$ – non-overlapping batches of $BS$ training samples selected using stratified sampling do
        perform a small number (20-30) of L-BFGS-B iterations on samples form $b$ starting at $\theta$ and update $\theta$
    end for
end for
```

Fig. 3. Repeated L-BFGS-B application to large subsets of training data used to train networks with nonnegative weights.

algorithm. Then, a step is taken along the gradient. The step length is set by the learning rate, which decreases over the run of the algorithm (annealing constants *B* and *C* had to be chosen for each experiment). The gradient step could set some weights to negative values. Nonnegativity is regained by the projection onto the feasible set, which sets negative weights to zero. Thus, after each training loop iteration the weights remain nonnegative.

Optimal performance of the stochastic gradient descent procedure requires tuning the learning rate magnitude and annealing schedule [34]. Faster convergence can often be obtained by estimating and using the curvature of the loss surface. The L-BFGS-B quasi-Newton function minimization algorithm [35] is often used for large-scale function minimization and it can accommodate nonnegativity constraints. It uses gradients computed during a few previous iterations to estimate the inverse of the Hessian at the current step. L-BFGS-B has to be provided with a precise value for the gradient of the loss function being minimized. Therefore, it is impossible to use in an online (stochastic) mode in which only a few training samples are processed at each step of the algorithm, resulting in noisy gradient estimates. For smaller data sets L-BFGS-B can be used in batch mode, computing the derivative of loss on the whole training data at each step. On large data sets the use of the whole set to compute the gradient can be prohibitively expensive. In this case training time can be reduced by dividing the data into large batches of several thousand training samples. Then, a specified amount of L-BFGS-B iterations is made on each batch of training samples [36]. This procedure is given in Fig. 3.

We have observed similar performance of both stochastic projected gradient descent and repeated L-BFGS-B runs on large subsets of training data. The second approach is slightly easier to use because it does not require the selection of the learning rate nor of its annealing schedule. Thus, we report experimental results using the L-BFGS-B approach.

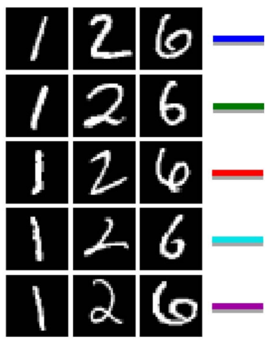
III. Experiments

The proposed approach needs the specification of the network's architecture and of five parameters, which control the regularization and the steepness of the sigmoid, λ. The number of hidden neurons was chosen to yield good classification accuracy while keeping the network reasonably small. For the reduced MNIST and Reuters data the networks have 10 and 15 hidden neurons, respectively, which allows easy inspection. For the full MNIST data, the number of hidden neurons had to be increased to 150, which hinders network interpretability. However, the hidden weights can be easily inspected visually when they are presented as images. For maximum interpretability, the hidden neurons should resemble threshold gates with only two states: ON and OFF. In reality, their output is squashed by the logistic sigmoid into the range (0, 1). To force the output of hidden neurons to be close to the limits of this range, the parameter λ was in all cases gradually increased. To determine the value of other parameters, we have first trained the network without regularization. We have then tested a few values of each of the regularization parameters that gave a similar value of the log-likelihood and regularization terms in (1). In Table I, we give the values used for the experiments. We note that the imposition of weight nonnegativity does not interfere with standard methods of choosing a proper network size and training parameters. The reader is referred to textbooks for further references on this important topic [4].

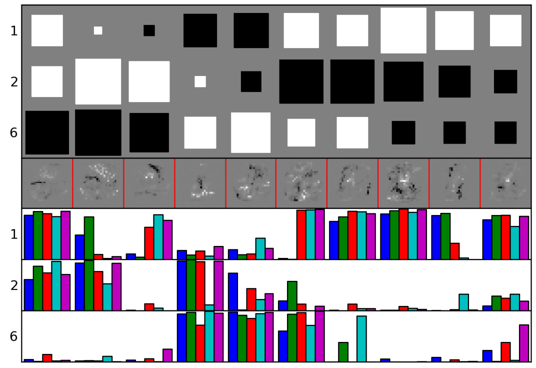
TABLE I
Values of Regularization Parameters Used in the Experiments

	Red. MNIST		Full MNIST		Reuters	
No. hidden	10	10	150	150	15	15
No. output	3	3	10	10	10	10
Non-neg?	F	T	F	T	F	T
λ	Annealed exponentially from 1.0 to 2.0					
p_{H1}	1e-4	3e-4	0	1e-5	1e-4	1e-4
p_{H2}	0	0	0	0	0	0
p_{C1}	0	0	0	1e-4	3e-4	3e-4
p_{C2}	1e-4	1e-4	1e-4	3e-6	3e-4	3e-4
p_S	0	0	0	1e-3	0	0

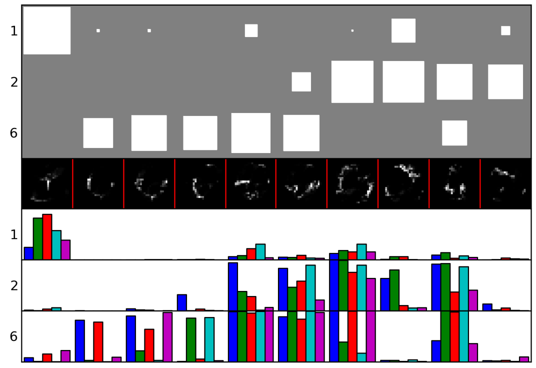
In the first experiment, we compared networks constructed with and without nonnegativity weight constraints on a subset of the MNIST handwritten digit data limited to digits 1, 2, and 6. The full MNIST data set contains 60 000 training and 10 000 testing grayscale images of handwritten digits, which were scaled and centered inside a 28×28 pixel box. It can be obtained along with a summary of classifiers' accuracies from http://yann.lecun.com/exdb/mnist/index.html. In this experiment small networks with 10 hidden neurons were used. We present a selection of test patterns and the weights of the two networks in Fig. 4. An immediate consequence of the nonnegativity constraints is sparsification of weights in the classification layer. Furthermore, the patterns learned by the hidden neurons allow easy interpretation. They are localized and tend to look like parts of digits (e.g., the images of weights of neurons in the columns 2–4 of the fourth row of Fig. 4 look like the rounded bottom of digit 6). In contrast, the hidden neurons of the unconstrained network are less localized. They contain both positive and negative weights covering most of the input image, which makes it harder to visualize to what patterns they respond. The bar charts indicate



(a)

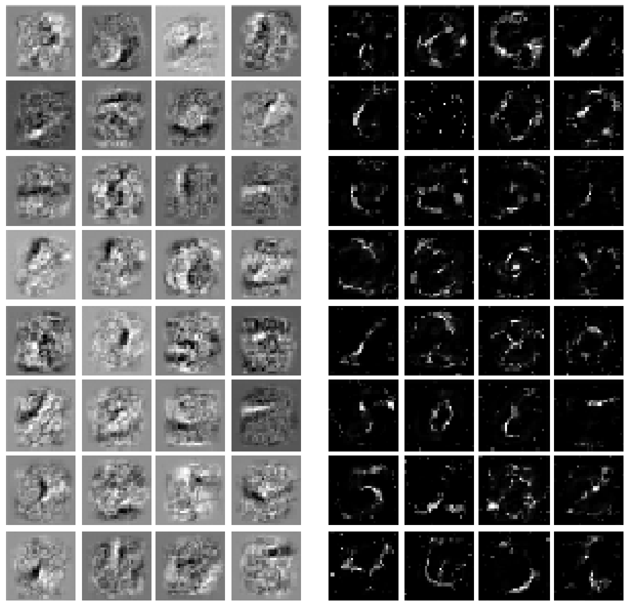


(b)

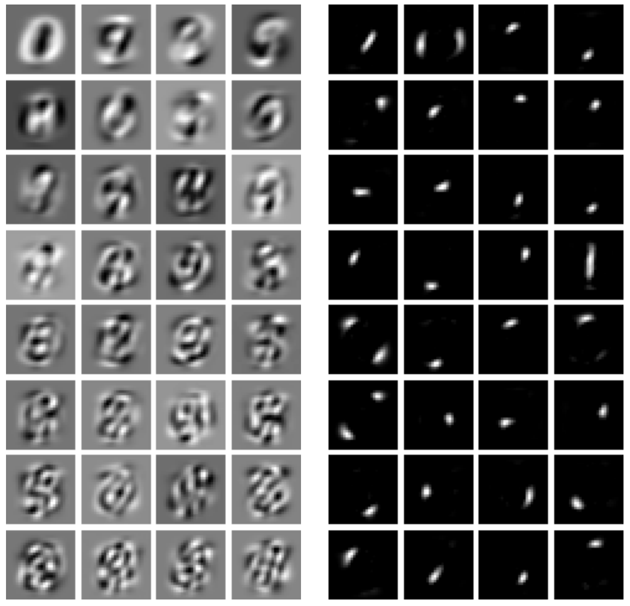


(c)

Fig. 4. (a) Exemplary digits from the MNIST data set. The weights of a network trained (b) without constraints and (c) with nonnegative constraints. The weights of the classification (output) layer are plotted as a diagram with one row for each output neuron and one column for every hidden (input) neuron. The area of each square is proportional to the weight's magnitude; white indicates positive and black negative sign. Below each column of the diagram, the weights of hidden neurons are printed as an image. The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with, the value 0 corresponding to gray in (b) and to black in (c). The biases are not shown. The hidden neurons have been rearranged for better presentation. The bar charts at the bottom of the plots show the activation of hidden neurons for the digits presented in (a). Each row shows the activations of each hidden neuron for five color-coded examples of the same digit.



(a) (b)



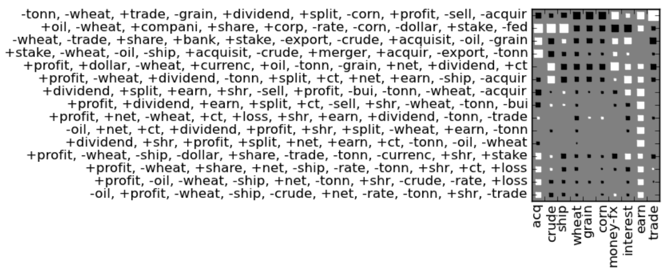
(c) (d)

Fig. 5. Weights of randomly selected 32 out of 150 hidden neurons of (a) unconstrained network and (b) network with weight nonnegativity constraints. (c) 32 first principal components. (d) 32 filters learned by NMF.

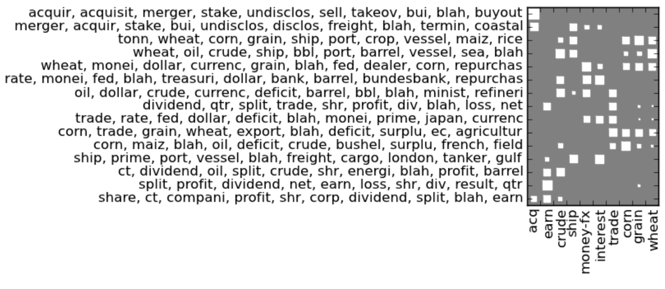
the activations of hidden neurons for the sample input patterns. It can be seen that neurons in both networks discriminate between digits and tend to work in the nonlinear parts of their activation functions, resembling threshold gates. The unconstrained network is more accurate and achieves 1% error rate, compared with 1.5% for the constrained one. In general, the trend was observed that constrained networks have shown lower testing accuracy. We believe that in certain situations a better insight into the data outweighs the benefits of an accurate but opaque classifier.

In the next large-scale experiment, we used the full MNIST data to build a constrained and unconstrained neural network with 150 hidden neurons and 10 outputs. We compare in Fig. 5, the depictions of weights of 32 randomly selected hidden neurons with 32 features obtained with PCA and with 32 obtained with NMF. Full networks are shown in the supplementary materials. The unconstrained network shows a much lower error rate of 2.4%, compared with 4.9% for the constrained one. To put those numbers into perspective, state-of-the-art 1998 error rate on the MNIST for a two layer neural network was 4.7%. Once again, the nonnegativity constraints have resulted in the emergence of sparser and more localized weight distributions of the hidden neurons, which often filter distinctive parts of digits. In contrast, the hidden neurons of the unconstrained network react to whole pictures, thus it is difficult to estimate intuitively their influence on the classifier's output. Similarly, the patterns learned by PCA are holistic, nonlocalized ones. But for the first few, it is hard to describe their contents. It is also difficult to see how they relate to the shapes of different digits. Furthermore, the NMF has learned sparse, localized, and interpretable features. However, only a few patterns resemble parts of digits, like the vertical bar (shown in the last column of Fig. 5). Most of the features seem to down-sample input images on a nonuniform grid and do not provide cues for classification. This is caused by two factors. First, unlike the neural network with nonnegative constraints on weights, the NMF model does not aim at class discrimination. Second, NMF imposes no limits on the number of features activated by a sample. Increasing the rank of factorization (the total number of features) only worsens the issue, as in the limit the identity matrix is a trivial NMF factor. In this case each NMF feature will be a single pixel and the NMF-transformed data will not be more useful for data analysis. On the other hand, decreasing the rank leads the NMF features to look like blurred shapes of the simplest digits. The addition of a constraint on the number of coactive features, while allowing a large total number of features, has been shown to promote the learning of a parts-based decomposition [29], [37], [38]. This is because, in contrast to a limited-rank decomposition, we assume a large dictionary of features which, in turn, must be complex enough to provide adequate input reconstruction from just the few active ones.

In the last experiment, we compared the networks on the Reuters-21578 text categorization collection. It is composed of documents that appeared in the Reuters newswire in 1987. We used the ModApte split limited to 10 most frequent categories. We have used a processed (stemming, stop-word removal) version in bag-of-words format obtained from http://people.kyb.tuebingen.mpg.de/pgehler/rap/. This data set is challenging because the borders between topics are fuzzy and documents may belong to many categories simultaneously. During training such documents were used with all possible labels. For testing a document was counted as correctly classified when the network assigned it to one of the classes to which it belonged. The networks had 15 hidden and 10 output neurons (one for each category). The unconstrained network is slightly more accurate and achieves an error rate of 12.4%, compared with 12.8% for the constrained one. The weights of the two networks are portrayed in Fig. 6. We provide an interpretation of the hidden neurons by listing words associated with the strongest weights. The word blah has no meaning and is artificially added noise. The nonnegative network has been observed to be more sensitive to it, as many hidden neurons react to it. The neurons in the unconstrained network seem to convey meaning by being both active and inactive, because the words associated with positive and negative weights fall into distinct categories. Furthermore, the matrix of output weights is dense and difficult to interpret.



(a)



(b)

Fig. 6. Networks trained on the Reuters-21578 data with (a) unconstrained weights and (b) nonnegative weight constraints. Input neurons are characterized by listing 10 words connected to weights having large absolute value. The $+$ and $-$ signs indicate the sign of the weight in (a). Each column of the diagram shows weights of an output neuron, the size varies with weight value and black or white filling indicates sign as in Fig. 4. The neurons have been rearranged for better presentation.

On the other hand, the output weights of the nonnegative network are sparse and allow for an interpretation of relations between topics. The closeness of topics corn, grain, and wheat is detected as the weights for those categories form a cluster. The topic trade is linked to categories describing goods that can be traded. The words listed for hidden neurons corroborate those interpretations, e.g., the neuron shown in the ninth row of Fig. 6 reacts to words trade, rate, fed, and dollar is linked through the classification weights matrix to topics money-fx (foreign exchange), interest, and trade.

A. Experiment Running Times

We have compared the time necessary to train networks with and without the nonnegativity constraints. The constrained networks required about twice as many epochs (passes through the full training set) to converge. However, a single epoch took only slightly more time for the constrained networks. This may be due to the fact that we have used the same L-BFGS-B implementation for both cases, imposing improper constraints $(-\infty,\infty)$ for the unconstrained networks. On the largest data set considered, the full MNIST benchmark, the unconstrained network required three epochs, which took 15 min on our machine. In contrast, the network with nonnegative weights required six epochs, which took 37 min.

IV. Conclusion

We have analyzed how constraining the weights of a neural network to be nonnegative affects network's accuracy and interpretability. Especially, on the Reuters text corpus the network's understandability has improved, as hidden neurons often responded to closely related groups of words, while only a small decrease in accuracy was observed. Therefore, the nonnegativity constraint can be used to enhance network transparency.

Appendix

Proof of the Shattering Property of Networks With Nonnegative Weights

We will show that a three layer network can shatter every combination of points. First, we show that adding a constant to all output weights does not change the value of the softmax function. Let the output layer compute the function

$$L_C = \text{SoftMax}(W \cdot L_H + B) \tag{3}$$

where L_C is the vector of classification (output) layer activations, W is the weight matrix, L_H is the vector of hidden layer activations, and B is the vector of bias values. Let $\mathbf{1}$ denote matrix whose all elements are 1 and let a be a constant. Then

$$\begin{aligned}\text{SoftMax}\,((a\mathbf{1} + W)L_H + B)\\ = \text{SoftMax}(a\mathbf{1} \cdot L_H + W \cdot L_H + B).\end{aligned} \tag{4}$$

For simplicity, substitute $C = W \cdot L_H + B$. The product $\mathbf{1} \cdot L_H$ is a vector whose all elements are equal to the sum of L_H. It follows that:

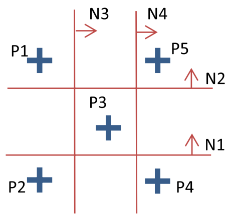
$$\begin{aligned}\text{SoftMax}(a\mathbf{1}L_H + C)_i &= \frac{\exp(\sum L_H)\exp(C_i)}{\sum_{j=1}^{n}\exp(\sum L_H)\exp(C_j)}\\ &= \text{SoftMax}(C)_i.\end{aligned} \tag{5}$$

Hence, we can transform any network with negative weights in the output layer into one with only nonnegative weights by adding a large enough constant.

We will construct a three layer (two hidden layers with sigmoid activation function followed by a classification layer with softmax activation) that will shatter a given set of m points described by their position in a n-dimensional space. Let $X \in \mathbb{R}^{n\times m}$ be the data matrix. We will show how to construct a network returning any labeling of those points. For simplicity we will assume that the gains in the logistic sigmoid transfer functions are infinite and the hidden neurons activations are always 0 or 1.

There are $O(nm)$ neurons in the first hidden layer. For every input dimension we first project all data points onto this dimension, then we select at most m threshold values between the projections. We add hidden neurons with a single nonzero weight equal to 1 corresponding to this dimension and bias equal to the threshold. Unless two columns of X are the same (i.e., two points are at exactly the same position), the activations H_1 of the first hidden layer are unique binary vectors. A 2-D example is shown in Fig. 7. Two threshold values are selected for each dimension. Each of the thresholds corresponds to a cut through the data plane and is implemented a single neuron. The weights of neurons are given—they are binary and uniquely identify each point.

There are m neurons in the second hidden layer, one for each point. Their weights are equal, the weight $W_{i,j}$ connecting the jth neuron in the first hidden layer to the ith neuron in the



	P1	P2	P3	P4	P5
N1	1	0	1	0	1
N2	1	0	0	0	1
N3	0	0	1	1	1
N4	0	0	0	1	1

Fig. 7. Construction of a network with nonnegative weights.

second one equals to $2j$. Thus, if we treat the first hidden layer activations as binary numbers, the products $W \cdot H_1$ are their decimal values. To every point corresponds one such number and we can order the points according to them. If the second hidden layer bias values are set to values in-between those numbers, the activations of the second hidden layer create a full-rank binary matrix (if the points are reordered, then for the kth point the k first neurons are active, while the $m - k$ remaining ones are zero. Hence, the activation matrix is triangular). Thus, we can solve for output weights for every possible labeling of data points. In the last step, we add a constant to output weights to ensure that all are nonnegative.

Acknowledgment

The authors would like to thank the anonymous reviewers for their many insightful comments and a careful proofing of this manuscript.

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees* (Statistics/Probability Series), vol. 19, C. Hall, Ed. Cleveland, OH, USA: CRC, 1984.

[2] J. R. Quinlan, *C4.5: Programs for Machine Learning*, M. B. Morgan, C. Leyba, and J. Hammett, Eds. Burlington, MA, USA: Morgan Kaufmann, 1992.

[3] W. W. Cohen, "Fast effective rule induction," in *Proc. 12th ICML*, 1995, pp. 1–9.

[4] J. M. Zurada, *Introduction to Artificial Neural Systems*. Eagan, MN, USA: West Publishing Company, 1992.

[5] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowl.-Based Syst.*, vol. 8, no. 6, pp. 373–389, Dec. 1995.

[6] W. Duch, R. Setiono, and J. Zurada, "Computational intelligence methods for rule-based data understanding," *Proc. IEEE*, vol. 92, no. 5, pp. 771–805, May 2004.

[7] B. Baesens, R. Setiono, C. Mues, and J. Vanthienen, "Using neural network rule extraction and decision tables for credit-risk evaluation," *Manag. Sci.*, vol. 49, no. 3, pp. 312–329, Mar. 2003.

[8] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999.

[9] A. J. Bell and T. J. Sejnowski, "The 'independent components' of natural scenes are edge filters," *Vis. Res.*, vol. 37, no. 23, pp. 3327–3338, Dec. 1997.

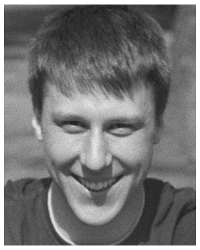
[10] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Netw.*, vol. 13, nos. 4–5, pp. 411–430, 2000.

[11] P. Paatero, "Least squares formulation of robust non-negative factor analysis," *Chemometrics Intell. Lab. Syst.*, vol. 37, no. 1, pp. 23–35, May 1997.

[12] D. T. Jones, "Protein secondary structure prediction based on position-specific scoring matrices," *J. Molecular Biol.*, vol. 292, no. 2, pp. 195–202, Sep. 1999.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[14] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
[15] R. Reed, “Pruning algorithms—A survey,” *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.
[16] Y. Le Cun, J. Denker, S. Solla, R. Howard, and L. Jackel, “Optimal brain damage,” in *Proc. NIPs*, vol. 2. 1990, pp. 598–605.
[17] B. Hassibi, D. Stork, and G. Wolff, “Optimal brain surgeon and general network pruning,” in *Proc. IEEE Int. Conf. Neural Netw.*, Jun. 1993, pp. 293–299.
[18] G. Castellano, A. Fanelli, and M. Pelillo, “An iterative pruning algorithm for feedforward neural networks,” *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 519–531, Jan. 1997.
[19] P. Bartlett, “The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network,” *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 525–536, 1998.
[20] A. Krogh and J. A. Hertz, “A simple weight decay can improve generalization,” in *Proc. NIPs*, 1991, pp. 950–957.
[21] G. Gnecco and M. Sanguineti, “Regularization techniques and suboptimal solutions to optimization problems in learning from data,” *Neural Comput.*, vol. 22, no. 3, pp. 793–829, Nov. 2009.
[22] M. Ishikawa, “Structural learning with forgetting,” *Neural Netw.*, vol. 9, no. 3, pp. 509–521, Apr. 1996.
[23] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *J. R. Statist. Soc., Series B (Statist. Methodol.)*, vol. 67, no. 2, pp. 301–320, Apr. 2005.
[24] S. Nowlan and G. Hinton, “Simplifying neural networks by soft weight-sharing,” *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992.
[25] R. Setiono, “Extracting M-of-N rules from trained neural networks,” *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 512–519, 2000.
[26] T. Huynh and J. Reggia, “Guiding hidden layer representations for improved rule extraction from neural networks,” *IEEE Trans. Neural Netw.*, vol. 22, no. 2, pp. 264–275, Feb. 2011.
[27] J. Chorowski and J. M. Zurada, “Extracting rules from neural networks as decision diagrams,” *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 2435–46, Dec. 2011.
[28] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. 25th Int. Conf. Mach. Learn.*, New York, NY, USA, 2008, pp. 1096–1103.
[29] M. Y. Ranzato, L. Boureau, and Y. LeCun, “Sparse feature learning for deep belief networks,” in *Proc. NIPS*, 2007, pp. 1185–1192.
[30] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. Le-Cun, “Learning invariant features through topographic filter maps,” in *Proc. IEEE Conf. CVPR*, Jun. 2009, pp. 1605–1612.
[31] A. Lemme, R. F. Reinhart, and J. J. Steil, “Efficient online learning of a non-negative sparse autoencoder,” in *Proc. ESANN*, 2010, pp. 1–6.
[32] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford Univ., 1995.
[33] P. Calamai and J. Moré, “English projected gradient methods for linearly constrained problems,” *English Math. Program.*, vol. 39, no. 1, pp. 93–116, 1987.
[34] Y. LeCun, L. Bottou, G. Orr, and K. Müller, “Efficient backprop,” *Neural Netw., Tricks Trade*, vol. 1524, no. 3, pp. 9–48, 1998.
[35] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization,” *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, Dec. 1997.
[36] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Ng, and Q. V. Le, “On optimization methods for deep learning,” in *Proc. 28th ICML*, 2011, pp. 265–272.
[37] B. Olshausen and D. Field, “Sparse coding with an overcomplete basis set: A strategy employed by V1?” *Vis. Res.*, vol. 37, no. 23, pp. 3311–3325, 1997.
[38] P. O. Hoyer, “Non-negative matrix factorization with sparseness constraints,” *J. Mach. Learn. Res.*, vol. 5, pp. 1457–1469, Aug. 2004.



Jan Chorowski (S’12) received the M.Sc. degree in electrical engineering from the Wrocław University of Technology, Wrocław, Poland, and the Ph.D. degree from the University of Louisville, Louisville, KY, USA,

He has been an Assistant Professor with the Department of Mathematics and Computer Science, University of Wroclaw, Wroclaw, since 2013. His current research interests include the development of machine learning algorithms, in particular, using neural networks.

Dr. Chorowski was the recipient of the University Scholarship from the University of Louisville.

Jacek M. Zurada (M’82–SM’83–F’96–LF’14) received the M.S. (Hons.) and Ph.D. (Hons.) degrees in electrical engineering from the Technical University of Gdansk, Gdansk, Poland, in 1968 and 1975, respectively.

He was the Department Chair from 2004 to 2006 and has been a Professor with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, USA, since 1989.

Dr. Zurada has served the profession and the IEEE in various elected capacities, including as the President of the IEEE Computational Intelligence Society from 2004 to 2005. In 2013, he served as the IEEE Technical Activities Chair-Elect and Chair of the IEEE TAB Periodicals Review and Advisory Committee. He is a Distinguished Speaker of the IEEE Computational Intelligence Society. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, PART I and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, PART II, and served on the editorial board of the PROCEEDINGS OF IEEE. From 1998 to 2003, he was the Editor-in-Chief of the IEEE TRANSACTIONS ON NEURAL NETWORKS. He is an Associate Editor of *Neural Networks, Neurocomputing, Schedae Informaticae*, and the *International Journal of Applied Mathematics and Computer Science*, the Advisory Editor of the *International Journal of Information Technology and Intelligent Computing*, and the Editor of the *Natural Computing Book Series* (Springer).