Constrained Autoencoders for Data Representation and Dictionary Learning

Babajide O. Ayinde and Jacek M. Zurada, *Life Fellow, IEEE*

***Abstract*—Without proper choice of constraints, autoencoders are capable of learning identity mapping or over-complete representations. The features learned by this architecture may be local, isolated or primitive. The extraction of features, however, can be controlled by judiciously enforcing some desired attributes, in form of constraints on its parameters. This paper gives an overview of autoencoders and such constraints for data representation. It also puts the autoencoder learning in a broader context of dictionary learning.**

***Index Terms*—Sparse autoencoder, part-based representation, deep learning, receptive field, denoising.**

I. Introduction

CONSTRAINED feature learning (CFL) is an important concept in feature engineering. It can unearth representation of data useful for such machine learning tasks as classification or compression. This latent representation could reveal what is important in data for a given discriminative task [1]. CFL heuristics that enable feature extraction can generate latent codes for test set during inference [1], [2]. CFL algorithms that range from sparse coding concept originally introduced in [3] to neural networks that implement learning with special constraints and work as feature extractors have become important tools in paradigm of representation learning.

These heuristics learn constrained representation usually by learning some dictionary terms that represent the data. The dictionary term is often used for semantic analysis such as document categorization. When dealing with other tasks and data, the terms are called receptive fields, filters, basis vectors or latent factors.

II. Dictionary Learning

Dictionary learning is best illustrated through sparse coding or data matrix factorization. Assume data matrix $\mathbf{X}$ contains m data vectors $\mathbf{x}_j$ as columns, each with n elements as shown in Fig. 1. Sparse coding aims to find a set of k basis vectors (columns ϕ_i of matrix $\Phi \in \mathbb{R}^{n\times k}$) and encodings (columns $\mathbf{a}_j$ of matrix $\mathbf{A} \in \mathbb{R}^{k\times m}$) such that $\mathbf{X} \approx \Phi\mathbf{A}$ for $\mathbf{X} \in \mathbb{R}^{n\times m}$, and $\mathbf{a}_j$ is a sparse vector for every j. When no limitation is imposed on k, it is possible to find via sparse coding an over-complete representation of data in which the number of basis vectors k is greater than the original data dimensionality n. That is, if $k > n$ the linear system of equations is under-determined and sparsity enforcement is needed to avoid obtaining a trivial solution [2].

In order to coerce $\mathbf{a}_j$ to be sparse for every j, a

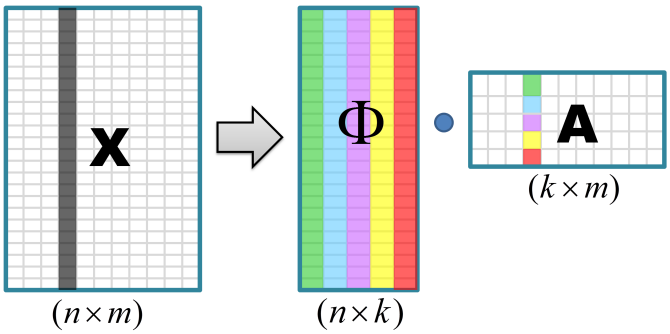


Fig. 1: Illustration of Data Matrix Factorization ($\mathbf{X} \approx \Phi\mathbf{A}$). $\mathbf{X}$ is the data matrix, columns of Φ are basis vectors, and columns of $\mathbf{A}$ are the encodings of the samples.

sparsity term is introduced in the objective function. Sparse combination of basis from an over-complete dictionary to represent data has been suggested as the mechanism with which mammal primary visual cortex (V1) work [3–5].

The data matrix decomposition is usually formulated as an optimization problem solvable by balancing out the error of approximation of $\mathbf{X}$ by $\Phi\mathbf{A}$ and the sparsity of $\mathbf{A}$. During the optimization process, a trivial solution may result in which entries of $\mathbf{A}$ are small due to sparsity enforcement but are compensated by allowing entries of Φ to assume large values [4], [6], [7]. To alleviate this problem, magnitude constraints are usually placed on the basis vectors ϕ_i through a process known as regularization by adding decay term to the objective function. This magnitude constraint is sometimes referred to as a weight decay penalty. Most sparse coding methods [3], [4], [8] require solving iterative optimization problem in order to compute feature descriptor which is usually computational expensive [2]. The complete optimization objective is thus formulated as in (1).

$$\min_{\mathbf{A},\Phi} \sum_{j=1}^{m} \left[\|\Phi\mathbf{a}_j - \mathbf{x}_j\|_2^2 + \gamma_1 \text{Sparsity}(\mathbf{a}_j) \right] + \gamma_2 \sum_{i=1}^{k} \|\phi_i\|_2^2 \tag{1}$$

B. O. Ayinde is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA. (e-mail: babajide.ayinde@louisville.edu).

J. M. Zurada is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA. (Corresponding author, e-mail: jacek.zurada@louisville.edu).

This work was supported in part by the NSF under grant 1641042.

where γ_1 and γ_2 are positive constants that adjust the relative importance of sparsity and magnitude (or regularization) constraints, respectively. Formula (1) minimizes the distance between the data and its representation given the learned basis.

III. Dictionary Learning through Constrained Autoencoders

One of the popular approaches to CFL is to train autoencoder (AE) in ways that enforces some desired attributes. The motivation behind the autoencoding is to reconstruct the input from its encoded representation with features that represent the data [9]. The reconstruction is usually achieved by additive linear (sometimes nonlinear) combination through decoding filters. After training, generating latent encodings for test samples is extremely fast, requiring a simple matrix-vector multiplication.

The model of the neural network AE shown in Fig. 2 which aims to reconstruct its input vector using unsupervised learning is given in (2).

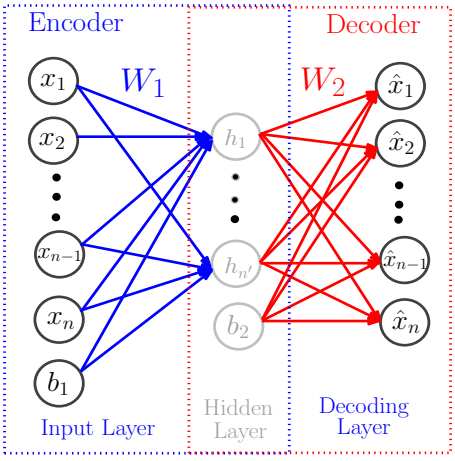


Fig. 2: Schematic diagram of a three-layer AE

$$\hat{\mathbf{x}} = f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \approx \mathbf{x} \tag{2}$$

where x is a normalized input vector, $\mathbf{W} = \{\mathbf{W}_1, \mathbf{W}_2\}$, and $\mathbf{b} = \{\mathbf{b}_1, \mathbf{b}_2\}$ respectively represent the weight and biases of the network. It is worth mentioning that the weight matrix $\mathbf{W}_2$ may optionally be constrained by $\mathbf{W}_2 = \mathbf{W}_1^T$, in which case the autoencoder is linear and said to have tied weights. Input data $\mathbf{X}$ is first encoded through $\mathbf{W}_1$ into features $\mathbf{h}$. In turn, features $\mathbf{h}$ are mapped back to the data $\hat{\mathbf{X}}$ through $\mathbf{W}_2$ in accordance with $\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1)$ where $\sigma(\cdot)$ is the activation function. One of the commonly used activation functions is the logistic sigmoid given as $\sigma(\mathbf{x}) = {}^1/_{(1+exp(-\mathbf{x}))}$. For the purpose of finding the parameters $\mathbf{W}$ and $\mathbf{b}$ in (2), the average reconstruction error serves as the optimization objective

$$\mathcal{J}_{AE}(\mathbf{W},\mathbf{b}) = \frac{1}{m}\sum_{i=1}^{m} \|\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2) - \mathbf{x}_i\|_2^2 \tag{3}$$

and corresponds to the first term in brackets of (1).

We should note that the dictionary learning (1) and AE learning (3) differ by two aspects. Firstly, the reconstruction error (3) involves mapping of data into itself by two matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, while the same error being the first term of (1) involves one matrix Φ. Secondly, (1) is solved by optimization, while (3) is based on unsupervised learning of $\mathbf{h}$.

Imposing meaningful limitations on network parameters generally forces AE network to learn representations that attempts to unearth the underlying structure in data. One of such limitations could be limiting the hidden layer size for compressed representation of the input. In this context, constrained AE implies that some constraints such as sparsity, nonnegativity, weight-decay regularization, and/or other constraint types are imposed on the learned features. Examples of such constraints are sparsity as in the Sparse Autoencoder (SAE) [10], or nonnegativity and sparsity as in Nonnegativity-Constrained Autoencoder (NCAE) [11–13].

Sparsification of features that represent data is increasingly important in learning, especially from big data. This is because sparsity can facilitate efficient and automatic feature selection. In addition, regularization can shrink the magnitude of AE weights and improve the generalization. Therefore, constrained AEs are not only used for feature dimensionality reduction, but also for extracting sparse features and to enhance data understanding.

Deep networks (DNs) based on AEs are created by stacking pretrained AEs layer by layer, followed by a supervised fine-tuning. They are able to extract salient features from input data through greedy, unsupervised, layerwise training algorithm. In deep autoencoding, cascade of AEs is trained to detect feature hierarchies from training samples to generate latent encodings. Each additional layer of AE adds an additional abstract representation of the input. Deep AE architectures invariably result in lower layerwise reconstruction error and a better representation of the input [14]. One of the key factors that contributes to lower error is the appropriate initialization achieved by pretraining each layer.

In vision-related task, basis vectors sensitive to a region in an image and to specific stimuli are called receptive fields (RFs). Fig. 3 illustrates the idea of constrained RF using L_1/L_2 Nonnegativity Constrained Sparse Autoencoder (L_1/L_2-NCSAE) [13] trained on synthetic data that comprises of three images as depicted. L_1/L_2-NCSAE is a specialized AE architecture capable of extracting nonnegative features (and nonnegative RFs) as shown. (L_1/L_2-NCSAE is explained in more detail in Section II). Input $\mathbf{X} \in \mathbb{R}^{25\times 3}$ consists of three 5×5 images. The three RFs are rows of weight matrix $\mathbf{W}_1 \in \mathbb{R}^{3\times 25}$. For visualization they are resized to match the square input image (both the inputs and the 25 weights of hidden neurons are presented as images). Neurons' outputs are the Activation Scores computed as the dot product of each RF and the input pattern.

It can be observed from Fig. 3 that first RF (1^{st} row of Activation Scores table) is most sensitive to first T-

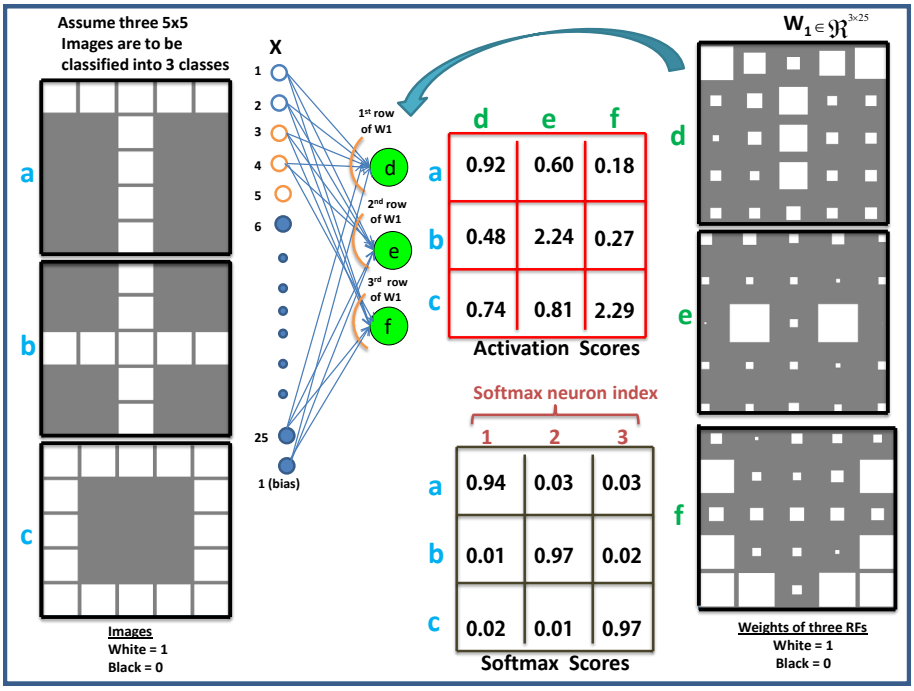


Fig. 3: Illustration of constrained (non-negative) RF feature extraction using a L_1/L_2-NCSAE trained on synthetic data with 3 images (left). The RFs learned (right) are rescaled and portrayed as images. The range of weights are scaled to [-1,1] and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. That is, black pixels indicate negative, and white pixels indicate positive weights. The dot product of each RF and input pattern shown as Activation Scores and the outputs of Softmax layer as Softmax scores. a,b, and c are the indices of input images; d,e, and f are the RFs indices. The biases are not shown.

shaped image and captures features that strongly react to T-shaped image. Similarly, second RF reacts mostly to the second input pattern. Likewise the third input pattern stimulates the third RF and maximally activates it with largest magnitude. It is remarked that using appropriate bias and softmax layer, first RF helps in classifying first image, second RF for classifying second one, and lastly, the third RF for third image. This observation is consistent with what is observed at the output of the softmax neurons given as Softmax scores in Fig. 3. The Softmax layer is also known as the classification or output layer [15–17]. Softmax scores are computed as $\text{Softmax}(\mathbf{W}_C \cdot \mathbf{h} + \mathbf{b}_C)$, where $\text{Softmax}(\mathbf{v})$ maps a vector $\mathbf{v}$ into a vector of values according to $\text{Softmax}(\mathbf{v})_i = \exp(\mathbf{v}_i)/\sum_{j=1}^{c} \exp(\mathbf{v}_j)$, $\mathbf{W}_C$ and $\mathbf{b}_C$ are respectively the matrix of weights and vector of bias values for the classification layer, and c is the size of the output vector equal to the number of classes.

In deep autoencoding, AE detects hierarchy of features that are useful in automatically extracting important features from inputs. Each added AE layer adds one more abstract representation of inputs, until producing a cascade of encodings [18], [19]. The generalization abilities of AEs deteriorates if specific measures such as sparsity, dropout, de-noising, and contraction of layers are not in place [20]. This problem of over-determined representation of data mapping into the DL multilayer architectures can be handled by training layers under specific sets of constraints.

The concept of RFs is not restricted only to visual information but also to many pattern recognition tasks such as those involving audio and semantic data.

A. *Constrained Autoencoders for Sparse Representation*

In AE settings, a network is considered over-sized if the size of the hidden layer is the same or larger than the input vector size n. In this scenario, AE can be forced to learn useful representation if additional constraints are added. These constraints can come in form of regularization to ensure sparsity of the hidden-layer representation or addition of noise in the hidden layer. Sparse representation can provide a interpretation of the input data in terms of a reduced number of parts and by extracting the hidden structure.

In order to force AE to learn sparse representation, $\mathbf{h}$ is bounded using the Kullback-Leibler (KL) divergence function [21–24]. If $h_j(\mathbf{x}_i)$ denotes the activation (or output) of hidden neuron j due to the input $\mathbf{x}_i$, the average activation of this particular neuron is given as:

$$\hat{\mathbf{p}}_j = \frac{1}{m} \sum_{i=1}^{m} h_j(\mathbf{x}_i) \tag{4}$$

If a sparse AE with target activation p is considered, one common method for imposing sparsity is to limit the ac-

tivation of hidden units using the KL function [10] as in (5)

$$\text{Sparsity}(p||\hat{\mathbf{p}}) = \sum_{j=1}^{n'} p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j} \quad (5)$$

One of many functions a regularizer provides is enforcing certain properties on the weights. Note that weight decay term is also added to the cost function of AE as to prevent overfitting [25]. For a conventional sparse autoencoder (SAE) the decay term is given as in (6).

$$\text{Decay}(w) = \frac{\alpha}{2} \sum_{l=1}^{2} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} ||w_{i,j}^{(l)}||_2^2 \quad (6)$$

where α is the weight penalty factor, and $w_{i,j}^{(l)}$ represents the connection between ith neuron in layer $l-1$ and jth neuron in layer l. The overall cost function based on (1) for SAE using penalization then becomes [10]:

$$J_{SAE}(\mathbf{W}, \mathbf{b}) = J_{AE}(\mathbf{W}, \mathbf{b}) + \beta \text{Sparsity}(p||\hat{\mathbf{p}}) + \text{Decay}(w) \quad (7)$$

where β controls the sparsity penalty term.

Other popular methods for sparsifying AE features while preventing overfitting are the dropout technique [26] and family of k-sparse AEs [27], [28]. In dropout technique, units and their connections are randomly dropped from the network during training. In effect, dropout tends to prevent neurons from co-adapting thereby leading to good generalization. The concept of k-sparse AE relies on identifying the k neurons with largest activations and setting the rest to zero to prevent overfitting. The k-sparse AE has been found suitable for many dataset because the value k can be tuned to obtain desirable sparsity level in conformity with each dataset.

B. Constrained Autoencoders for Part-based Data Representation

Part-based representation is a way of decomposing data into parts, which when additively combined regenerate the data [29]. Also, drawing inspiration from the idea of Nonnegative Matrix Factorization (NMF) and sparse coding [29], [30], the hidden structure of data can be unfolded by learning features that have capabilities to extract the data parts. NMF enforces the encoding of both the basis vectors and features to be nonnegative thereby resulting in additive data representation. Similar to the data decomposition illustrated in Fig. 1, NMF decomposes data matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ with nonnegative real entries into product of two nonnegative matrices $\mathbf{W} \in \mathbb{R}^{n \times k}$ and $\mathbf{H} \in \mathbb{R}^{k \times m}$, that is, $\mathbf{X} \approx \mathbf{WH}$. Incorporating sparse coding within NMF for the purpose of encoding data is computationally expensive, while with AEs, this incorporation is learning-based and fast. In addition, the performance of AE can be enhanced using NCAEs with part-based data representation capability [31–33].

As shown in [29], one way of representing data is by shattering it into various distinct pieces in a manner that additive merging of these pieces can reconstruct the original data. Mapping this intuition to AEs, the idea is to sparsely disintegrate data into parts in the encoding layer and additively process the parts to recombine the original data in the decoding layer. This is achieved by imposing nonnegativity constraint in form of a penalty term and this can be achieved by replacing the Decay term in (7) with (8) to yield NCAE [11].

$$\text{Decay}(w) = \begin{cases} \frac{\alpha}{2} \sum_{l=1}^{2} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(w_{ij}^{(l)}\right)^2 & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \quad (8)$$

where $\alpha > 0$ is a nonnegativity-constraint weight penalty factor.

Also, Nonnegative Sparse Autoencoder (NNSAE) trained with an online algorithm and tied weights and linear output activation function is capable of extracting nonnegative features for part-based representation of data [34]. In NNSAE, asymmetric weight decay function was used to constrain network parameters to be nonnegative. The weight decay term in (6) for SAE can also be viewed as imposing Gaussian prior distribution on network weights while NNSAE uses a weight decay mechanism that assumes a virtually deformed Gaussian prior that is skewed with respect to the sign of the weight. For the purpose of illustrating part-based data decomposition, NCAE and NNSAE networks were trained on the 60,000 training set of MNIST digit data and both AEs

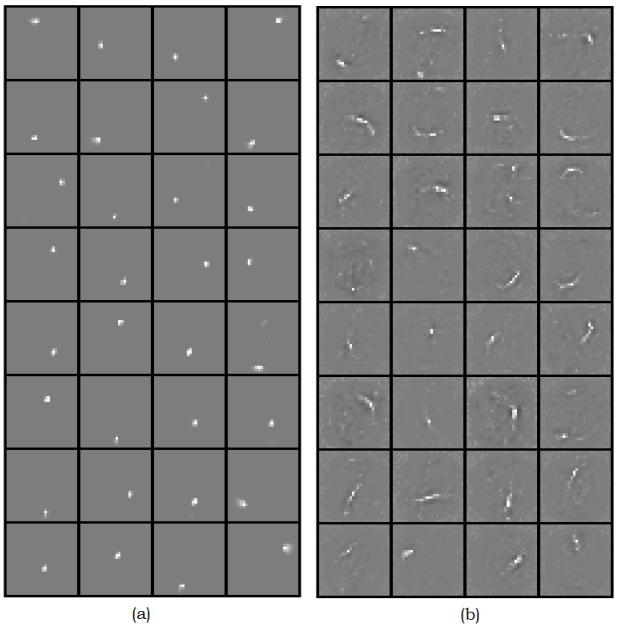


Fig. 4: RFs or weights of randomly selected 32 out of 196 ($n' = 196$) hidden neurons of (a) NNSAE (b) NCAE trained using MNIST dataset. Black pixels indicate negative, grey pixels indicate zero-valued weights and white pixels indicate positive weights. The range of weights are scaled to [-1,1] and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color.

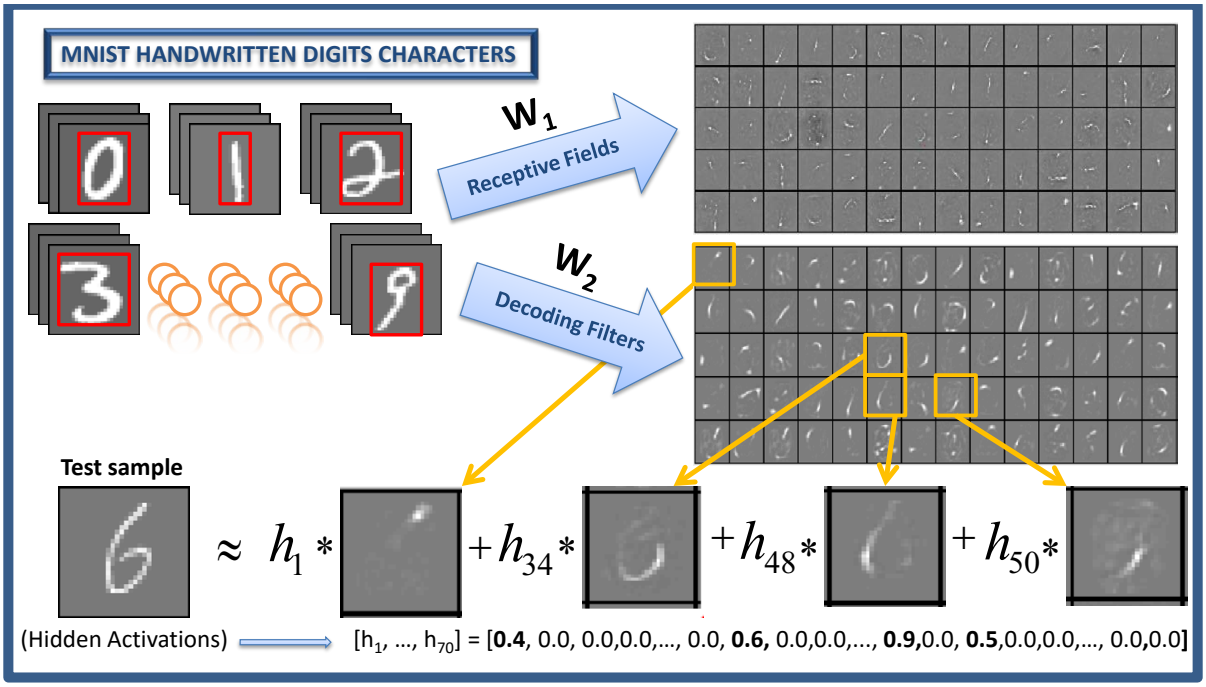


Fig. 5: Representation of test image as a linear combination of 4 out of 196 constrained RFs and decoding filters learned from MNIST dataset using NCAE with linear output activation function. Input consist of 784 values corresponding to a 28×28 pixel image. Only 70 RFs with largest activations to test image "6" and their corresponding decoding filters are shown. The RFs and the decoding filters are rescaled and portrayed as images on the right hand side. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to [-1,1] and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color. The biases are not shown.

have 196 hidden neurons. Their weights are portrayed as images of RFs. Figs. 4a and b show the receptive fields learned by NNSAE and NCAE, respectively. It can be observed that the RFs learned select structures of handwritten digits such as strokes and dots. The learned featured are localized and tend to look like parts of digits.

Part-based representation is illustrated in Fig.5 using NCAE trained on MNIST handwritten characters. NCAE with linear decoder architecture (that is, the activation function $\sigma(.)$ for decoding layer is identity function) was trained in such a manner that the column of $\mathbf{W}_1$ are coerced to be sparse. RFs and the decoding filters are displayed on the right hand side. A test image of digit 6 shown is filtered through the network and activations **h** are listed. The vector of activations is very sparse since it only stimulates 4 of RFs. The test sample can be reconstructed by additively combining four outputs of decoding filters scaled with magnitudes of select **h** values.

C. Constrained Autoencoders for Enhanced Data Understanding

Neural networks are known for building hierarchical models but do so in twisted ways that are difficult to understand [17]. Attempts have been made to reconcile interpretability of autoencoding network [13] and solve a long-standing open problem of understanding their processing. The emergence of part-based representation can be conceptually tied to the nonnegativity constraints. Borrowing this concept from biological analogies, one way to foster the understandability in autoencoding is to constrain the encoder and decoder's weights to be nonnegative. This allows easier human interpretation since the cancelation of terms in the scalar products summation is eliminated [17]. In practice, the cancelations are discouraged rather than eliminated in order to ensure a good accuracy. Enhanced data understanding can result when the input data can be decomposed into parts in each layer, while the weights are constrained to be nonnegative and sparse as shown in [13]. This imposition of nonnegativity constraint can be incorporated into the cost function of SAE in (7) in form of a penalty term and the resulting L_1/L_2-NCSAE [12], [13]. The decay term in (8) is replaced with (9). It must be noted that the constraint imposed using (9) is a soft one, hence, some weights will still be negative. However, the number and magnitudes of these negative weights are reduced.

$$\text{Decay}(w) = \sum_{l=1}^{2} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \begin{cases} \alpha_1 \Gamma(w_{ij}^{(l)}, \kappa) + \frac{\alpha_2}{2} ||w_{ij}^{(l)}||^2 & w_{ij} < 0 \\ 0 & w_{ij} \geq 0 \end{cases} \tag{9}$$

where α_1 and α_2 are nonnegativity-constraint weight penalty for L_1 and L_2 terms, respectively.

Although (9) is capable of enforcing nonnegativity in AE, however, its L_1 norm term is non-differentiable at the origin,

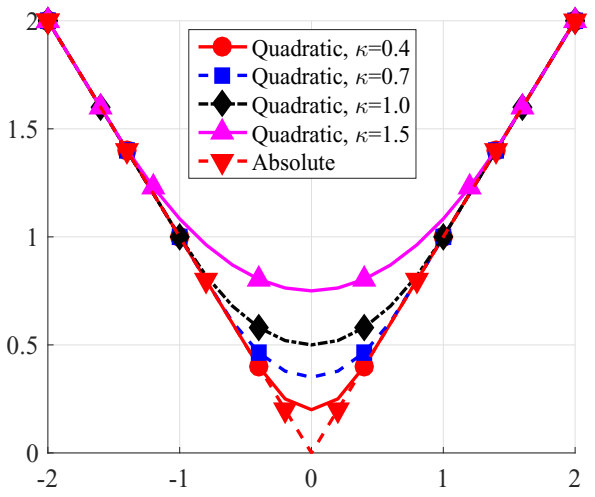


Fig. 6: Absolute function approximation using quadratic smoothing functions with parameter κ = 0.4, 0.7, 1.0 and 1.5. κ tunes the approximation of L_1 norm.

and this can lead to numerical instability during simulations. To circumvent this drawback, smoothing functions that approximate L_1 norm is used [13]. One of the well known smoothing functions is the quadratic function given in (10) and depicted in Fig. 6. Given any finite dimensional vector $\mathbf{z}$ and positive constant κ, the following smoothing function approximates L_1 norm:

$$\Gamma(\mathbf{z},\kappa) = \begin{cases} ||\mathbf{z}|| & ||\mathbf{z}|| > \kappa \\ \dfrac{||\mathbf{z}||^2}{2\kappa} + \dfrac{\kappa}{2} & ||\mathbf{z}|| \leq \kappa \end{cases} \tag{10}$$

and its gradient is:

$$\nabla_{\mathbf{z}}\Gamma(\mathbf{z},\kappa) = \begin{cases} \dfrac{\mathbf{z}}{||\mathbf{z}||} & ||\mathbf{z}|| > \kappa \\ \dfrac{\mathbf{z}}{\kappa} & ||\mathbf{z}|| \leq \kappa \end{cases} \tag{11}$$

To demonstrate this concept, a subset of NORB normalized-uniform dataset [35] with class labels four-legged animals, human figures, airplanes was extracted. The full data set consists of $24,300$ training images and $24,300$ test images of 50 toys from 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The training and testing sets comprise 5 instances of each category. Each image consists of two channels, each of size 96x96 pixels. The inner 64x64 pixels of each channel were taken and resized using bicubic interpolation to 32x32 pixels that form a vector with 2048 entries as the input. The 2048-10-3 network was trained on the subset of the NORB data set. Fig. 7a shows the randomly sampled test patterns and the weights of the output layer are given in Fig. 7b. The AE was using the L_1/L_2-NCSAE with 10 hidden neurons and a softmax layer with 3 neurons. It can be observed in Fig. 7b that sparsification of the output layer weights is one of the aftermaths of the nonnegativity constraints imposed on the network. In addition, the patterns learned by neurons in each layer are localized, and this allows easy interpretation of isolated data parts. The patterns visually show nonnegative weights covering most of the image of the input object making it easier to visualize to what patterns they respond [12].

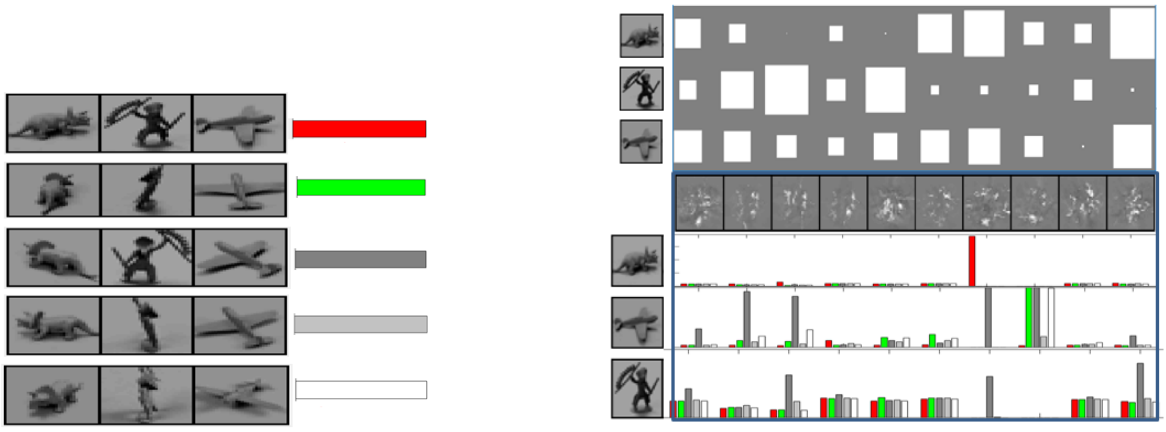
The bar charts indicate the activations of hidden neurons for the sample input patterns. It can be seen that neurons in the network discriminate among objects in the images and react differently to input patterns. In contrary to the general observation that networks constrained for interpretability show lower testing accuracy, the L_1/L_2 nonnegativity constraint improves the AE network interpretation while maintaining comparable testing accuracy.

D. Other Constraints in Autoencoders

Another useful property of AEs is robustness to partial destruction of the input, that is, partially corrupted inputs produce almost the same representation [36]. For example, high dimensional redundant data usually are recoverable from partial dimensions of the data. This is because data contains stable structures that depend on a combination of many dimensions. Imposition of the above constraint on AE result in what is well known as Denoising Autoencoder (DAE) [14]. This constraint can facilitate DAE to improve the robustness of its representation through reconstructing a clean input from a corrupted one. Training a DAE is similar to training a basic AE.

Generally, for the input $\mathbf{x}$, some input components are randomly selected and their values forced to 0, while others remain unchanged. Thus, a corrupted version $\tilde{\mathbf{x}}$ is obtained. $\tilde{\mathbf{x}}$ is then mapped to a hidden representation $\mathbf{h}$ from which $\hat{\mathbf{x}}$ is reconstructed. The key difference between DAE and basic AE is that $\hat{\mathbf{x}}$ of DAE (given as $\hat{\mathbf{x}} = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\tilde{\mathbf{x}}+\mathbf{b}_1)+\mathbf{b}_2)$) is a function of $\tilde{\mathbf{x}}$ as opposed a function of $\mathbf{x}$ for AE (given as $\hat{\mathbf{x}} = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2)$).

Constraint can also be in form of convolution where repetitive features are discovered and weights are shared across the entire input field. AEs that utilize this concept are referred to as Convolutional AEs (CAEs). One of the main difference between fully connected AE and CAE is that the latter shares weights among adjacent locations in the input in order to preserve spatial locality [37]. Thus, CAE typically results in some degree of shift, scale, and distortion invariance. The concept of weight sharing in convolutional network renders a way of reducing the number of trainable parameters and has been shown to have regularizing effect. Typically in convolutional neural network, a layer is stimulated by a set of units located in the local neighborhood in the previous layer to extract localized features. Subsequent layers then combine these feature to extract higher level features.



(a) (b)

Fig. 7: (a) Example images from NORB data set and (b) The weights trained using L_1/L_2-NCSAE. The weights of the softmax layer are plotted as a diagram at topmost layer in (b). Each row of the plot corresponds to each output neuron and each column for every hidden neuron. The magnitude of the weight corresponds to the area of each square; white indicates positive and black negative sign. Underneath the plot are the receptive fields learned from the reduced NORB dataset. 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. The biases are not shown. The activations of hidden neurons for the NORB objects presented in (a) are depicted on the bar chart at the bottom of the plot. Each row shows the activations of each hidden neuron for five color-coded examples of the same object.

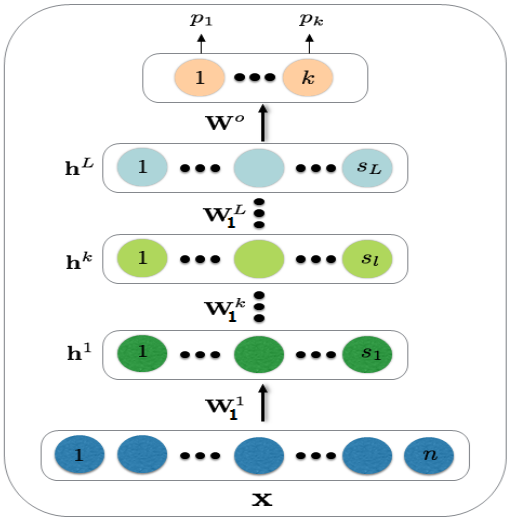


Fig. 8: Schematic diagram of Stacked Constrained Autoencoders (SCAE) plus Softmax Classifier (SMC)

E. Deep High-Level Feature Learning with Constrained Autoencoders

To facilitate multilevel feature extraction and increased mapping accuracy, several AEs can be stacked into a deep architecture as depicted in Fig. 8 and trained with the input of a layer being the activation of its preceding layer. The activations of the last AE are then used usually as the input to either a linear regression layer (if the output values are continuous) or logistic regression layer (if the output is discrete). Finally, in the fine-tuning stage, the weights of all the layers are tuned simultaneously in a supervised fashion to improve the classification accuracy [23]. To illustrate this concept, two L_1/L_2-NCSAEs were stacked and trained as described above on 150 samples of 1, 2 and 3 digits from MNIST handwritten dataset, 30 samples from each digit category.

The number of hidden neurons was chosen to obtain reasonably good classification accuracy while keeping the network reasonably small. For this subset of MNIST data, the hidden sizes of the first and second AEs were chosen to be 10 to allow easy inspection. The network is intentionally kept small because the full MNIST data would require larger hidden layer size and this may limit network interpretability. The filtering of a test image of the digit 2 is shown in Fig. 9. It can be seen that the fourth and seventh receptive fields of the first AE layer have dominant activations (with activation values 0.12 and 0.13 respectively) and they capture most information about the test input. Also, they are able to filter distinct part of input digit. The outputs of the first layer sigmoid constitute higher level features extracted from test image with emphasis on the fourth and seventh features. Subsequently in second layer the second, sixth, eight, and tenth neurons have dominant activations (with activation values 0.0914, 0.0691, 0.0607, and 0.0606 respectively) because they have stronger connections with the dominant neurons in first layer than the rest. Lastly in the softmax layer, the second neuron was 99.62% activated because it has strongest connections with the dominant neurons in second layer thereby classifying the test image as ”2”.

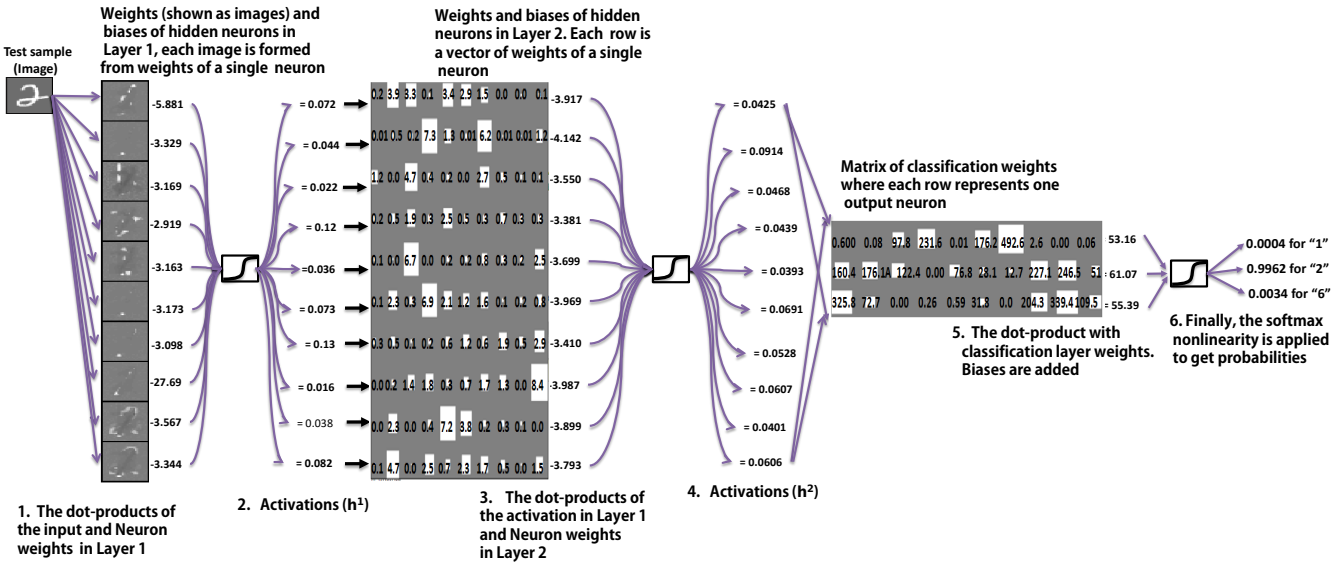


Fig. 9: Filtering the signal through two stacked L_1/L_2-NCSAEs trained using the reduced MNIST data set with class labels 1, 2 and 6. The test image is a 28×28 pixels image unrolled into a vector of 784 values. Both the input test sample and the RFs of the first AE layer are presented as images.The architecture is 784-10-10-3. The weights of the output layer are plotted as a diagram with one row for each output neuron and one column for every hidden neuron in 2^{nd} layer [13]. Black pixels indicate negative, and white pixels indicate positive weights. The range of weights are scaled to [-1,1] and mapped to the graycolor map. $w = -1$ is assigned to black, $w = 0$ to grey, and $w = 1$ is assigned to white color.

IV. Computational Consideration and Software Libraries

Many existing deep learning libraries have good AE implementations but their functionalities vary. The list of libraries in Table I is to highlight some of the most popular tools for experimenting with AE and is by no means exhaustive.

DeepLearn [38] as its name suggests is a DL library implemented in MATLAB. For very large data, some of the computational limitations can be overcome by GPU parallelization [39]. Theano is another optimized open-source symbolic tensor manipulation framework for developing machine/deep learning algorithms. It is very handy for implementing gradient-based methods such as DL models that repeatedly compute tensor-based mathematical expressions which can be easily coded in Theano and compiled on either a CPU or a GPU [47]. A good number of AEs, such as denoising AE [36], have efficient Theano implementation. Many libraries such as Keras and Pylearn2 use Theano as backend engine. Although Keras can also be configured to use TensorFlow backend by setting appropriate backend field to TensorFlow. Keras provides high-level building blocks for developing DL models and relies on well-optimized tensor manipulation libraries with GPU and CPU support. Pylearn2 offers implementation of a variety of models and training algorithms that form integral modules of DL.

Torch is another scientific computational framework built on Lua programming language [40], [41]. It supports GPU and comes with efficient CUDA backend and neural networks libraries [47]. AE architectures such as SAE, stacked SAE, denoising AE, and variational AEs also have Torch implementations.

Pytorch [42] is a Python-based deep learning framework that provides tape-based autograd system functionality. Not only does it provide standalone DL research platform, it can also serve to replace NumPy (a popular package for scientific computing with Python) in order to take advantage of the GPU implementation for speedup. PyTorch uses one of the fastest implementation of reverse-mode auto-differentiation which allows the dynamics of the network to be arbitrarily changed with little or overhead. Its memory usage is very efficient compared to its counterpart such as Torch which enables training of larger models.

TensorFlow is an open-source symbolic tensor manipulation framework developed by Google, Inc. It is one of the recently introduced DL framework based on C++ [43] with Python API. It uses data flow graphs to perform numerical computations where the nodes denote mathematical operations and the edges denote multidimensional data array communicated between them.

V. Concluding Remarks

This paper reviews data representation and latent feature extraction using several types of constrained autoencoders. In addition to sparsity, non-negative receptive fields can be enforced during learning to enhance interpretability of data and its additive properties where applicable. Comparisons are made between computing of matrix Φ with quasi-basis vectors as columns containing dictionary terms and

TABLE I: Deep Learning Libraries with Autoencoder Implementation.

Library	Language	Operating System	Architecture	Popular Optimizer	Loss Function	GPU-Enabled?
DeepLearn [38]	MATLAB	Windows Linux	SAE [a] SSAE [b] DAE [c]	BGD[d] SGD[e]	MSE[f]	Yes
Torch7 [40], [41]	Lua	Linux Android iOS	SAE SSAE DAE	SGD LBFGS [g] CG [h]	MSE Cross-Entropy	Yes
Pytorch [42]	Python	Linux OSX	SAE	SGD Adam Adamax Adagrad Adadelta	MSE Cross Entropy BCE [i] NLL [j] KLD [k] HE [l]	Yes
TensorFlow [43]	C++ (with Python API)	Linux Windows	SAE DAE	BGD SGD	MSE Cross-Entropy	Yes
Theano [44]	Python	Linux Mac	DAE	SGD	NGL[m]	Yes
Pylearn2 [45] (Built on Theano)	Python	Linux	SAE SSAE DAE	BGD SGD	MSE Cross-Entropy	Yes
Keras [46] (Built on Theano or TensorFlow)	Python	Linux	SAE CAE [n] DAE	SGD RMSprop Adagrad Adadelta	MSE Categorical Cross-Entropy	Yes

[a]Sparse Autoencoder
[b] Stacked Sparse Autoencoder
[c]Denoising Autoencoder
[d]Batch Gradient Descent
[e]Stochastic Gradient Descent
[f]Mean Square Error
[g]Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm
[h]Conjugate Gradient
[i]Binary Cross-Entropy
[j]Negative Log Likelihood
[k]Kullback-Leibler divergence
[l]Hinge Embedding
[m]Negative log-likelihood
[n]Convolutional Autoencoder

unsupervised learning of a cascade of two RFs being rows of encoding matrix $\mathbf{W}_1$ and columns of the decoding matrix $\mathbf{W}_2$.

Special constraints and regularization are capable of forcing AEs and their RFs to learn representations that can unearth the underlying structures in data. Such specialized limitations including enforcing sparse and nonnegative decompositions and dictionaries that can help shatter data into parts and in fostering the understanding of the hidden data structure. This, in turn, can facilitate informed decision making.

REFERENCES

[1] J. Snoek, R. P. Adams, and H. Larochelle, "Nonparametric guidance of autoencoder representations using label information," *Journal of Machine Learning Research*, vol. 13, no. Sep, pp. 2567–2588, 2012.

[2] K. Kavukcuoglu, R. Fergus, Y. LeCun *et al.*, "Learning invariant features through topographic filter maps," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1605–1612.

[3] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1," *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[4] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," *Advances in neural information processing systems*, vol. 19, p. 801, 2007.

[5] J. F. Murray and K. Kreutz-Delgado, "Learning sparse overcomplete codes for images," *The Journal of VLSI Signal Processing*, vol. 45, no. 1, pp. 97–110, 2006.

[6] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 689–696.

[7] J. Chorowski, "Review of dimensionality reduction techniques," *Technical Paper*, 2010.

[8] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural computation*, vol. 20, no. 10, pp. 2526–2563, 2008.

[9] J. Wang, H. He, and D. V. Prokhorov, "A folded neural network autoencoder for dimensionality reduction," *Procedia Computer Science*, vol. 13, pp. 120–127, 2012.

[10] A. Ng, "Sparse autoencoder," in *CS294A Lecture notes*. URL https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf: Stanford University, 2011.

[11] E. Hosseini-Asl, J. M. Zurada, and O. Nasraoui, "Deep learning of part-based representation of data using sparse autoencoders with

nonnegativity constraints," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 27, no. 12, pp. 2486–2498, 2016.

[12] B. O. Ayinde, E. Hosseini-Asl, and J. M. Zurada, "Visualizing and understanding nonnegativity constrained sparse autoencoder in deep learning," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2016, pp. 3–14.

[13] B. O. Ayinde and J. M. Zurada, "Deep learning of constrained autoencoders for enhanced understanding of data," *submitted to Neural Networks and Learning Systems, IEEE Transactions on*, June, 2016.

[14] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *25th International Conference on Machine Learning*. ACM, 2008, pp. 1096–1103.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[17] J. Chorowski and J. M. Zurada, "Learning understandable neural networks with nonnegative weight constraints," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 1, pp. 62–69, 2015.

[18] B. O. Ayinde and J. M. Zurada, "Nonredundant sparse feature extraction using autoencoders with receptive fields clustering," *submitted to Neural Networks*, September, 2016.

[19] B. Ayinde and J. Zurada, "Clustering of receptive fields in autoencoders," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 1310–1317.

[20] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1798–1828, 2013.

[21] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area v2," *Advances in Neural Information Processing Systems*, vol. 7, pp. 873–830, 2007.

[22] V. Nair and G. E. Hinton, "3d object recognition with deep belief nets," *Advances in Neural Information Processing Systems*, pp. 1339–1347, 2009.

[23] G. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[24] C. Poultney, S. Chopra, and Y. Cun, "Efficient learning of sparse representations with an energy-based model," *Advances in Neural Information Processing Systems*, pp. 1137–1144, 2006.

[25] J. Moody, S. Hanson, A. Krogh, and J. A. Hertz, "A simple weight decay can improve generalization," *Advances in Neural Information Processing Systems*, vol. 4, pp. 950–957, 1995.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[27] A. Makhzani and B. Frey, "K-sparse autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.

[28] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders," in *Advances in Neural Information Processing Systems*, 2015, pp. 2791–2799.

[29] 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, 1999.

[30] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[31] M. Ranzato, Y. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," *Advances in Neural Information Processing Systems*, vol. 20, pp. 1185–1192, 2007.

[32] S. J. Wright and J. Nocedal, *Numerical optimization*. Springer New York, 1999, vol. 2.

[33] T. D. Nguyen, T. Tran, D. Phung, and S. Venkatesh, "Learning partsbased representations with nonnegative restricted boltzmann machine," in *Asian Conference on Machine Learning*, 2013, pp. 133–148.

[34] A. Lemme, R. Reinhart, and J. Steil, "Online learning and generalization of parts-based image representations by non-negative sparse autoencoders," *Neural Networks*, vol. 33, pp. 194–203, 2012.

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

[36] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[37] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[38] R. B. Palm, "Deeplearn toolbox," 2014.

[39] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognition and Image Analysis*, vol. 26, no. 1, pp. 9–15, 2016.

[40] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[41] Torch. [Online]. Available: https://github.com/Kaixhin/Autoencoders

[42] Pytorch. [Online]. Available: http://pytorch.org

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[44] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, 2010, pp. 1–7.

[45] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio, "Pylearn2: a machine learning research library," *arXiv preprint arXiv:1308.4214*, 2013.

[46] F. Chollet, "Keras: Theano-based deep learning library," *Code: https://github. com/fchollet. Documentation: http://keras. io*, 2015.

[47] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," *arXiv preprint arXiv:1511.06435*, 2015.

Babajide Ayinde (S'09) received the M.Sc. degree in Engineering Systems and Control from the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. He is currently a Ph.D. candidate at the University of Louisville, Kentucky, USA and a recipient of University of Louisville fellowship. His current research interests include unsupervised feature learning and deep learning techniques and applications.

Jacek M. Zurada (M'82-SM'83-F'96-LF'14) Ph.D., has received his degrees from Gdansk Institute of Technology, Poland. He now serves as a Professor of Electrical and Computer Engineering at the University of Louisville, KY. He authored or co-authored several books and over 380 papers in computational intelligence, neural networks, machine learning, logic rule extraction, and bioinformatics, and delivered over 100 presentations throughout the world.

In 2014 he served as IEEE V-President, Technical Activities (TAB Chair). He also chaired the IEEE TAB Periodicals Committee, and TAB Periodicals Review and Advisory Committee and was the Editor-in-Chief of the IEEE Transactions on Neural Networks (1997-03), Associate Editor of the IEEE Transactions on Circuits and Systems, Neural Networks and of The Proceedings of the IEEE. In 2004-05, he was the President of the IEEE Computational Intelligence Society.

Dr. Zurada is an Associate Editor of Neurocomputing, and of several other journals. He has been awarded numerous distinctions, including the 2013 Joe Desch Innovation Award, 2015 Distinguished Service Award, and five honorary professorships. He has been a Board Member of IEEE, IEEE CIS and IJCNN.