Deep Sparse-coded Network (DSN)

Youngjune Gwon
Harvard University
gyj@eecs.harvard.edu

Miriam Cha
Harvard University
miriamcha@fas.harvard.edu

H. T. Kung
Harvard University
kung@harvard.edu

***Abstract*—We present Deep Sparse-coded Network (DSN), a deep architecture based on multilayer sparse coding. It has been considered difficult to learn a useful feature hierarchy by stacking sparse coding layers in a straightforward manner. The primary reason is the modeling assumption for sparse coding that takes in a dense input and yields a sparse output vector. Applying a sparse coding layer on the output of another tends to violate the modeling assumption. We overcome this shortcoming by interlacing nonlinear pooling units. Average- or max-pooled sparse codes are aggregated to form dense input vectors for the next sparse coding layer. Pooling achieves nonlinear activation analogous to neural networks while not introducing diminished gradient flows during the training. We introduce a novel backpropagation algorithm to finetune the proposed DSN beyond the pretraining via greedy layerwise sparse coding and dictionary learning. We build an experimental 4-layer DSN with the ℓ_1-regularized LARS and the greedy-ℓ_0 OMP, and demonstrate superior performance over a similarly-configured stacked autoencoder (SAE) on CIFAR-10.**

I. Motivation

Representational power of single-layer feature learning is limited for tasks that involve large complex data objects such as a high-resolution image of human face. Best current practices in visual recognition use deep architectures based on autoencoder [1], restricted Boltzmann machine (RBM) [2], and convolutional neural network (CNN) [3]. A deep architecture stacks two or more layers of feature learning units in the hope of discovering hierarchical representations for data. In other words, deep architectures allow us to understand a feature at each layer using the features of the layer below. Such hierarchical decomposition is particularly useful when it is hard to resolve ambiguity of the low-level (or localized) features of data. Deep architectures also promote representational efficiency. We can achieve compaction of all characteristic features for the entire image, book, or lengthy multimedia clip to a single vector.

In an empirical analysis by Coates, Lee and Ng [4], a simple single-layer scheme, which encodes features with basis vectors learned from K-means clustering, is found on par (or sometimes, even superior) to RBM, deep neural network, and CNN for classification tasks on the CIFAR-10 and NORB datasets. We have also been able to draw a similar conclusion from our experiments with sparse coding. With these in mind, it is sound to build a deep architecture on sparse coding. Unfortunately, this takes more than just stacking sparse coding units in layers. From sparse coding research on hierarchical feature learning [5], [6], we could deduce some explanations for the difficulty. First, sparse coding (in particular, the ℓ_1-regularized LASSO or LARS) is computationally expensive for multilayering and associated optimizations. From our experience, it is cumbersome to simply connect multiple sparse coding units and run data as a feedforward network. Secondly, sparse coding makes an inherent assumption on the input being non-sparse. Therefore, a straightforward approach to take the output from one sparse coding unit for an input to another is flawed. Lastly, it is difficult to optimize all layers of sparse coding jointly. One consensual notion of deep learning suggests layer-by-layer unsupervised pretraining should be followed by supervised finetuning of the whole system, which is commonly done by backpropagation.

In this paper, we propose Deep Sparse-coded Network (DSN), a deep architecture for sparse coding as a principled extension from its single-layer counterpart. We consider both the ℓ_1-regularized LASSO/LARS [7], [8] and greedy-ℓ_0 OMP [9] as a legitimate sparse coding method. Using max pooling as nonlinear activation analogous to neural networks, we avoid linear cascade of dictionaries and keep the effect of multilayering in tact. This architectural usage will remedy the problem of too many feature vectors by aggregating them to their maximum elements and help preserve translational invariance of higher-layer representations. Beyond the layer-by-layer pretraining, we propose a novel backpropagation algorithm that is specific to multilayer sparse coding interlaced by spatial max pooling. We have validated empirically that the proposed backpropagation algorithm can further optimize the classification performance by 4–6% gain in accuracy.

Rest of this paper is organized as follows. In Section 2, we provide a brief background on sparse coding. Section 3 will introduce DSN, explain its architectural principles, and discuss training algorithms. In Section 4, we present an empirical evaluation of DSN, and Section 5 concludes the paper.

II. Sparse Coding Background

Originally used to explain neuronal activations [10], sparse coding is an unsupervised method to learn an efficient representation of data using a small number of basis vectors. It has been used to discover high-level features present in data

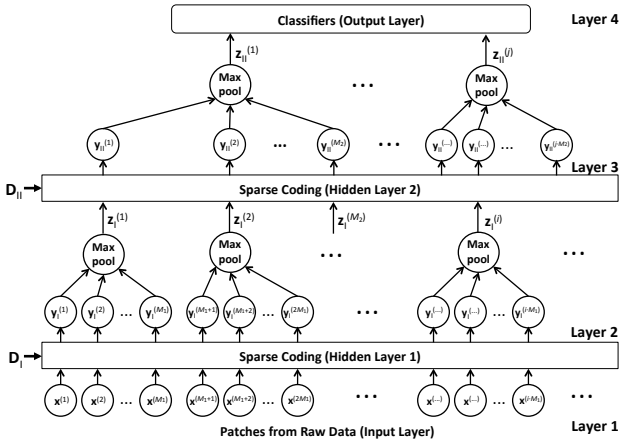

Fig. 1. Deep Sparse-coded Network (DSN) with four layers

from unlabeled examples. Given an example $\mathbf{x} \in \mathbb{R}^N$, sparse coding searches for a representation $\mathbf{y} \in \mathbb{R}^K$ (*i.e.*, the feature vector for $\mathbf{x}$) while simultaneously updating the dictionary $\mathbf{D} \in \mathbb{R}^{N \times K}$ of K basis vectors by

$$\min_{\mathbf{D},\mathbf{y}} \|\mathbf{x} - \mathbf{D}\mathbf{y}\|_2^2 + \lambda \|\mathbf{y}\|_1 \quad \text{s.t. } \|\mathbf{d}_i\|_2 \leq 1, \forall i \qquad (1)$$

where $\mathbf{d}_i$ is ith dictionary atom in $\mathbf{D}$, and λ is a regularization parameter that penalizes over the ℓ_1-norm, which induces a sparse solution. With $K > N$, sparse coding typically trains an overcomplete dictionary.

A more direct way to control sparsity is to regularize on the ℓ_0 pseudo-norm $\|\mathbf{y}\|_0$, describing the number of nonzero elements in $\mathbf{y}$. However, it is known to be intractable to compute the sparsest ℓ_0 solution in general. The approach in Eq. (1) is called least absolute shrinkage and selection operator (LASSO) [7], a convex relaxation of the ℓ_0 sparse coding that induces sparse $\mathbf{y}$'s. We use least angle regression (LARS) [8] to solve the LASSO problem. We also consider orthogonal matching pursuit (OMP) [9], a greedy-ℓ_0 sparse coding algorithm that computes an at-most S-sparse $\mathbf{y}$ extremely fast by

$$\min_{\mathbf{D},\mathbf{y}} \|\mathbf{x} - \mathbf{D}\mathbf{y}\|_2^2 \quad \text{s.t. } \|\mathbf{y}\|_0 \leq S. \qquad (2)$$

III. Deep Sparse-coded Network (DSN)

A. Notation

We denote input vector $\mathbf{x}$, its sparse code $\mathbf{y}$, and the pooled sparse code $\mathbf{z}$. An ith input vector (patch) is designated as $\mathbf{x}^{(i)}$. Sparse coding layers are the "hidden" layers of DSN. We use subscripted Roman numerals to indicate sparse coding layers. For example, $\mathbf{D}_{\mathrm{I}}$ means the first hidden layer's dictionary. Note that the first hidden layer is the overall layer 2. Accordingly, $\mathbf{y}_{\mathrm{I}}$ is the sparse code computed at the first hidden layer with the input $\mathbf{x}$ and $\mathbf{D}_{\mathrm{I}}$, and $\mathbf{z}_{\mathrm{I}}$ the pooled sparse code over multiple $\mathbf{y}_{\mathrm{I}}$'s.

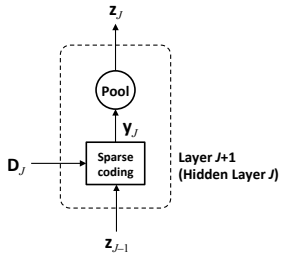

Fig. 2. DSN layering module

B. Architectural Overview

Deep Sparse-coded Network (DSN) is a feedforward network built on multilayer sparse coding. In Figure 1, we present an exemplar 4-layer DSN. This is a deep architecture since there are two hidden layers of sparse coding, each of which can learn corresponding level's feature representations and train own dictionary of basis vectors. Similar to neural network, layers 1 and 4 are the input and output layers. The input layer takes in vectorized patches drawn from the raw data, which will be sparse coded and max pooled, propagating up the layers. The output layer consists of classifiers or regressors specific to application needs.

Figure 2 depicts a stackable layering module to build DSN. Sparse coding and pooling units together constitute the module. The Jth hidden layer (for $J \geq$ II) takes in pooled sparse codes $\mathbf{z}_{J-1}$'s from the previous hidden layer and produces $\mathbf{y}_J$ using dictionary $\mathbf{D}_J$. Max pooling $\mathbf{y}_J$'s yields pooled sparse code $\mathbf{z}_J$ that are passed as the input for hidden layer $J+1$.

C. Algorithms

Hinton *et al.* [11] suggested *pretrain* a deep architecture with layer-by-layer unsupervised learning and finetune via *backpropagation*, a supervised algorithm popularized by neural network. We explain training algorithms for DSN using the architecture in Figure 1.

1) Pretraining via layer-by-layer sparse coding and dictionary learning: DSN takes in *spatially* contiguous patches from an image or *temporally* consecutive patches from time-series data to make the overall feature learning meaningful. Optimally, patches are preprocessed by normalization and whitening. The input layer is organized as pooling groups of M_1 patches: $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(M_1)}\}$, $\{\mathbf{x}^{(M_1+1)}, \mathbf{x}^{(M_1+2)}, \ldots, \mathbf{x}^{(2M_1)}\}$, $\cdots$. Sparse coding and dictionary learning at hidden layer 1 compute sparse codes $\mathbf{y}_{\mathrm{I}}^{(i)}$'s

while learning $\mathbf{D}_{\mathrm{I}}$ jointly

$$\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(M_1)}\} \xrightarrow{\mathbf{D}_{\mathrm{I}}} \{\mathbf{y}_{\mathrm{I}}^{(1)},\dots,\mathbf{y}_{\mathrm{I}}^{(M_1)}\}$$
$$\{\mathbf{x}^{(M_1+1)},\dots,\mathbf{x}^{(2M_1)}\} \xrightarrow{\mathbf{D}_{\mathrm{I}}} \{\mathbf{y}_{\mathrm{I}}^{(M_1+1)},\dots,\mathbf{y}_{\mathrm{I}}^{(2M_1)}\}$$
$$\vdots$$

Max pooling at hidden layer 1 aggregates multiple sparse codes

$$\{\mathbf{y}_{\mathrm{I}}^{(1)},\dots,\mathbf{y}_{\mathrm{I}}^{(M_1)}\} \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{I}}^{(1)}$$
$$\vdots$$

Hidden layer 1 passes the pooled sparse codes $\{\mathbf{z}_{\mathrm{I}}^{(1)},\mathbf{z}_{\mathrm{I}}^{(2)},\dots\}$ to hidden layer 2. Sparse coding and dictionary learning continue at hidden layer 2 using $\mathbf{z}_{\mathrm{I}}$'s as input

$$\{\mathbf{z}_{\mathrm{I}}^{(1)},\dots,\mathbf{z}_{\mathrm{I}}^{(M_2)}\} \xrightarrow{\mathbf{D}_{\mathrm{II}}} \{\mathbf{y}_{\mathrm{II}}^{(1)},\dots,\mathbf{y}_{\mathrm{II}}^{(M_2)}\}$$
$$\vdots$$

Pooling groups at hidden layer 2 consist of M_2 pooled sparse codes from hidden layer 1. Max pooling by M_2 yields

$$\{\mathbf{y}_{\mathrm{II}}^{(1)},\dots,\mathbf{y}_{\mathrm{II}}^{(M_2)}\} \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{II}}^{(1)}$$
$$\vdots$$

Pretraining completes by producing dictionaries $\{\mathbf{D}_{\mathrm{I}} \in \mathbb{R}^{N\times K_1}, \mathbf{D}_{\mathrm{II}} \in \mathbb{R}^{K_1\times K_2}\}$ and the highest hidden layer's pooled sparse codes $\{\mathbf{z}_{\mathrm{II}}^{(1)},\mathbf{z}_{\mathrm{II}}^{(2)},\dots\}$ with each $\mathbf{z}_{\mathrm{II}}^{(j)} \in \mathbb{R}^{K_2}$.

Max pooling is crucial for our DSN architecture. It reduces the total number of features by aggregating sparse codes to their max elements. More importantly, max pooling serves as a nonlinear activation function in neural network. Without *nonlinear* pooling, multilayering has no effect: $\mathbf{x} = \mathbf{D}_{\mathrm{I}}\mathbf{y}_{\mathrm{I}}$ and $\mathbf{y}_{\mathrm{I}} = \mathbf{D}_{\mathrm{II}}\mathbf{y}_{\mathrm{II}}$ implies $\mathbf{x} = \mathbf{D}_{\mathrm{I}}\mathbf{D}_{\mathrm{II}}\mathbf{y}_{\mathrm{II}} \approx \mathbf{D}\mathbf{y}_{\mathrm{II}}$ because linear cascade of dictionaries is simply $\mathbf{D} \approx \mathbf{D}_{\mathrm{I}}\mathbf{D}_{\mathrm{II}}$ regardless of total number of layers.

2) Training classifiers at output layer: DSN learns each layer's dictionary greedily during pretraining. The resulting highest hidden layer output $\mathbf{z}_{\mathrm{II}}$ is already a powerful feature for classification tasks. Suppose DSN output layer predicts a class label $\hat{l} = h_{\mathbf{w}}(\phi)$, where $h_{\mathbf{w}}(.)$ is a standard linear classifier or logistic/softmax regression that takes a feature encoding ϕ as input. Note that ϕ is encoded on $\mathbf{z}_{\mathrm{II}}$, but depends on DSN setup. For instance, we may have $\phi = [\mathbf{z}_{\mathrm{II}}^{(1)};\ \mathbf{z}_{\mathrm{II}}^{(2)};\ \mathbf{z}_{\mathrm{II}}^{(3)};\ \mathbf{z}_{\mathrm{II}}^{(4)}]$ if the highest hidden layer yields four pooled sparse codes per training example.

For simplicity, assume $\phi = \mathbf{z}_{\mathrm{II}}$. A DSN classifier then computes $\hat{l} = h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}}) = \mathbf{w}^{\top}\cdot\mathbf{z}_{\mathrm{II}} + w_0$. We train the classifier weight $\mathbf{w} = [w_1\ \dots\ w_{K_2}]^{\top}$ and the bias w_0 using labeled examples $\{(\mathbf{X}_1,l_1),\dots,(\mathbf{X}_m,l_m)\}$ in a supervised process by filling the input layer with patches from each example—the ith example $\mathbf{X}_i$ consists of $\{\mathbf{x}_i^{(1)},\mathbf{x}_i^{(2)},\dots\}$, where $\mathbf{x}_i^{(k)}$ is the kth patch—and working up the layers to compute $\mathbf{z}_{\mathrm{II}}$'s that are used to train the DSN classifiers.

3) Backpropagation: By now, we have the DSN output layer with trained classifiers, and this is a good working pipeline for discriminative tasks. However, we might further improve the performance of DSN by optimizing the whole network in a supervised setting. Is backpropagation possible for DSN?

DSN backpropagation is quite different from conventional neural network or deep learning architectures. We explain our backpropagation idea again using the example DSN in Figure 1. The complete feedforward path of DSN is summarized by

$$\mathbf{x} \xrightarrow{\mathbf{D}_{\mathrm{I}}} \mathbf{y}_{\mathrm{I}} \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{I}} \xrightarrow{\mathbf{D}_{\mathrm{II}}} \mathbf{y}_{\mathrm{II}} \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{II}} \overset{\text{classify}}{\longrightarrow} \hat{l}$$

We define the loss or cost function for the DSN classification

$$J(\mathbf{z}_{\mathrm{II}}) = \frac{1}{2}\left\|\hat{l}-l\right\|^2 = \frac{1}{2}\left\|h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}})-l\right\|^2 \tag{3}$$

Our objective now is to propagate the loss value down the reverse path and adjust sparse codes. Fixing classifier weights $\mathbf{w}$, we back-estimate optimal $\mathbf{z}_{\mathrm{II}}^*$ that minimizes $J(\mathbf{z}_{\mathrm{II}})$. To do so, we perform gradient descent learning with $J(\mathbf{z}_{\mathrm{II}})$ that adjusts each element of vector $\mathbf{z}_{\mathrm{II}}$ by

$$z_{\mathrm{II},k} := z_{\mathrm{II},k} - \alpha\frac{\partial J(\mathbf{z}_{\mathrm{II}})}{\partial z_{\mathrm{II},k}} \tag{4}$$

where α is the learning rate, $\mathbf{z}_{\mathrm{II}} = [z_{\mathrm{II},1}\ z_{\mathrm{II},2}\ \dots\ z_{\mathrm{II},K_2}]^{\top}$, and K_2 is the number of basis vectors in dictionary $\mathbf{D}_{\mathrm{II}}$ for hidden layer 2. Since an optimal $\mathbf{z}_{\mathrm{II}}^*$ is estimated by correcting $\mathbf{z}_{\mathrm{II}}$, the partial derivative is with respect to each element $z_{\mathrm{II},k}$

$$\frac{\partial J(\mathbf{z}_{\mathrm{II}})}{\partial z_{\mathrm{II},k}} = [h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}})-l)]\,\frac{\partial h(\mathbf{z}_{\mathrm{II}})}{\partial z_{\mathrm{II},k}} = [h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}})-l]\,w_k$$

Here, note our linear classifier $h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}}) = w_0 + w_1\cdot z_{\mathrm{II},1} + \dots + w_{K_2}\cdot z_{\mathrm{II},K_2}$. Therefore, the following gradient descent rule adjusts $\mathbf{z}_{\mathrm{II}}$ to $\mathbf{z}_{\mathrm{II}}^*$

$$z_{\mathrm{II},k} := z_{\mathrm{II},k} + \alpha\,[l - h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}})]\,w_k \tag{5}$$

This update rule is intuitive because it down-propagates the error $[l - h_{\mathbf{w}}(\mathbf{z}_{\mathrm{II}})]$ proportionately to the contribution from each $z_{\mathrm{II},k}$ and adjusts accordingly.

Using the corrected $\mathbf{z}_{\mathrm{II}}^*$, we can correct the unpooled original $\mathbf{y}_{\mathrm{II}}$'s to optimal $\mathbf{y}_{\mathrm{II}}^*$'s by a procedure called *putback* illustrated in Figure 3. At hidden layer 2, we have performed max pooling by M_2. For putback, we need to keep the original M_2 $\mathbf{y}_{\mathrm{II}}$'s that have resulted $\mathbf{z}_{\mathrm{II}}$ in memory so that corrected values at $\mathbf{z}_{\mathrm{II}}^*$ are *put back* to corresponding locations at the original sparse codes $\mathbf{y}_{\mathrm{II}}$'s and yield error-adjusted $\mathbf{y}_{\mathrm{II}}^*$'s.

With $\mathbf{y}_{\mathrm{II}}^*$, going down a layer is straightforward. By sparse coding relation, we just compute $\mathbf{z}_{\mathrm{I}}^* = \mathbf{D}_{\mathrm{II}}\mathbf{y}_{\mathrm{II}}^*$. Next, we do

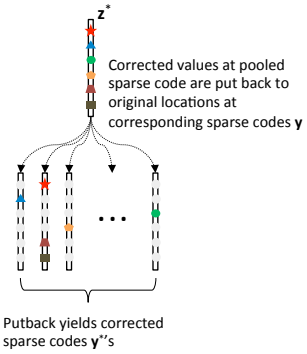

Fig. 3. Putback corrects sparse codes $\mathbf{y}$ from $\mathbf{z}^*$

another putback at hidden layer 1. Using $\mathbf{z}_{\mathrm{I}}^*$, we obtain $\mathbf{y}_{\mathrm{I}}^*$'s. Each pooling group at hidden layer 1 originally has M_1 $\mathbf{y}_{\mathrm{I}}$'s that need to be saved in memory.

With $\mathbf{y}_{\mathrm{I}}^*$'s, we should now correct $\mathbf{D}_{\mathrm{I}}$, not $\mathbf{x}$, because it does not make sense to correct given data input. Hence, down-propagation of the error for DSN stops here, and we up-propagate corrected sparse codes to finetune the dictionaries and classifier weights. We consider two methods to adjust basis vectors of the dictionaries. First method is to use rank-1 update in batches. The idea is to compute the residual matrix that isolates a contribution by each basis vector only. We update each basis vector iteratively with the first principle component computed via singular value decomposition of the residual (*i.e.*, the first column of matrix $\mathbf{U}$). This method is also used in the inner-loop of K-SVD [12].

Our second method uses online gradient descent. We define the loss function with respect to $\mathbf{D}_{\mathrm{I}}$

$$J(\mathbf{D}_{\mathrm{I}}) = \frac{1}{2} \left\| \mathbf{D}_{\mathrm{I}} \mathbf{y}_{\mathrm{I}}^* - \mathbf{x} \right\|_2^2 \tag{6}$$

Adjusting $\mathbf{D}_{\mathrm{I}}$ requires to solve the following optimization problem given examples $(\mathbf{x}, \mathbf{y}_{\mathrm{I}}^*)$

$$\min_{\mathbf{d}_{\mathrm{I},k}} J(\mathbf{D}_{\mathrm{I}}) \quad \text{s.t.} \quad \|\mathbf{d}_{\mathrm{I},k}\|_2^2 = 1 \quad \forall k \tag{7}$$

where $\mathbf{d}_{\mathrm{I},k}$ is the kth basis vector in $\mathbf{D}_{\mathrm{I}}$. Taking the partial derivative with respect to $\mathbf{d}_{\mathrm{I},k}$ yields

$$\frac{\partial J(\mathbf{D}_{\mathrm{I}})}{\partial \mathbf{d}_{\mathrm{I},k}} = (\mathbf{D}_{\mathrm{I}} \mathbf{y}_{\mathrm{I}}^* - \mathbf{x})\, y_{\mathrm{I},k}^*$$

where $\mathbf{y}_{\mathrm{I}}^* = \begin{bmatrix} y_{\mathrm{I},1}^* & \dots & y_{\mathrm{I},K_1}^* \end{bmatrix}^\top$ and $\mathbf{y}_{\mathrm{I}} = \begin{bmatrix} y_{\mathrm{I},1} & \dots & y_{\mathrm{I},K_1} \end{bmatrix}^\top$. We obtain the update rule to adjust $\mathbf{D}_{\mathrm{I}}$ by gradient descent

$$\mathbf{d}_{\mathrm{I},k} := \mathbf{d}_{\mathrm{I},k} - \beta (\mathbf{D}_{\mathrm{I}} \mathbf{y}_{\mathrm{I}}^* - \mathbf{x})\, y_{\mathrm{I},k}^* \tag{8}$$

We denote the corrected dictionary $\mathbf{D}_{\mathrm{I}}^*$. We redo sparse coding at hidden layer 1 with $\mathbf{D}_{\mathrm{I}}^*$ followed by max pooling. Similarly at hidden layer 2, we update $\mathbf{D}_{\mathrm{II}}$ to $\mathbf{D}_{\mathrm{II}}^*$ by

$$\mathbf{d}_{\mathrm{II},k} := \mathbf{d}_{\mathrm{II},k} - \gamma (\mathbf{D}_{\mathrm{II}} \mathbf{y}_{\mathrm{II}}^* - \mathbf{z}_{\mathrm{I}}^\dagger)\, y_{\mathrm{II},k}^* \tag{9}$$

where $\mathbf{z}_{\mathrm{I}}^\dagger$ is the pooled sparse code over M_1 $\mathbf{y}_{\mathrm{I}}^\dagger$'s from sparse coding redone with $\mathbf{D}_{\mathrm{I}}^*$. Using corrected dictionary $\mathbf{D}_{\mathrm{II}}^*$, we also redo sparse coding and max pooling at hidden layer 2. The resulting pooled sparse codes $\mathbf{z}_{\mathrm{II}}^\dagger$ are the output of the highest hidden layer, which will be used to retrain the classifier $h_{\mathbf{w}}$. All of the steps just described are a single iteration of DSN backpropagation. We run multiple iterations until convergence.

The corrections made during down-propagation for DSN backpropagation are summarized by

$$\mathbf{z}_{\mathrm{II}} \overset{\mathrm{GD}}{\longrightarrow} \mathbf{z}_{\mathrm{II}}^* \overset{\text{putback}}{\longrightarrow} \mathbf{y}_{\mathrm{II}}^* \overset{\mathbf{D}_{\mathrm{II}}}{\longrightarrow} \mathbf{z}_{\mathrm{I}}^* \overset{\text{putback}}{\longrightarrow} \mathbf{y}_{\mathrm{I}}^*.$$

The corrections by up-propagation follow

$$\mathbf{D}_{\mathrm{I}} \overset{\mathrm{GD}}{\longrightarrow} \mathbf{D}_{\mathrm{I}}^* \overset{\mathrm{SC}}{\longrightarrow} \mathbf{y}_{\mathrm{I}}^\dagger \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{I}}^\dagger \overset{\mathrm{GD}}{\longrightarrow} \mathbf{D}_{\mathrm{II}}^* \overset{\mathrm{SC}}{\longrightarrow} \mathbf{y}_{\mathrm{II}}^\dagger \overset{\text{max pool}}{\longrightarrow} \mathbf{z}_{\mathrm{II}}^\dagger \overset{\mathrm{GD}}{\longrightarrow} h_{\mathbf{w}}$$

where GD stands for gradient descent, and SC sparse coding. We present the backpropagation algorithm for general L-layer DSN in Algorithm 1.

Algorithm 1 DSN backpropagation

require Pretrained $\{\mathbf{D}_{\mathrm{I}}, \mathbf{D}_{\mathrm{II}}, \dots, \mathbf{D}_{L-2}\}$ and classifier $h_{\mathbf{w}}$
input Labeled training examples $\{(\mathbf{X}_1, l_1), \dots, (\mathbf{X}_m, l_m)\}$
output Fine-tuned $\{\mathbf{D}_{\mathrm{I}}^*, \mathbf{D}_{\mathrm{II}}^*, \dots, \mathbf{D}_{L-2}^*\}$ and classifier $h_{\mathbf{w}}^*$

repeat
  subalgorithm Down-propagation
    for J := $L-2$ **to** I
      if J == $L-2$
        Compute classifier error $\epsilon^{(i)} = l^{(i)} - h_{\mathbf{w}}(\mathbf{z}_{\mathrm{J}}^{(i)})$ $\forall i$
        Compute $\mathbf{z}_{\mathrm{J}}^{*(i)}$ by $z_{\mathrm{J},k}^{(i)} := z_{\mathrm{J},k}^{(i)} + \alpha \cdot \epsilon^{(i)} \cdot w_k$ $\forall i, k$
      else
        Compute $\mathbf{z}_{\mathrm{J}}^{*(i)} = \mathbf{D}_{\mathrm{J}+1} \mathbf{y}_{\mathrm{J}+1}^{*(i)}$ $\forall i$
      end
      Estimate $\mathbf{y}_{\mathrm{J}}^{*(i)}$ from $\mathbf{z}_{\mathrm{J}}^{*(i)}$ via putback $\forall i$
    end
  end
  subalgorithm Up-propagation
    for J := I **to** $L-2$
      if J == I
        Compute $\mathbf{D}_{\mathrm{I}}^*$ by Eq. (8)
        Compute $\mathbf{y}_{\mathrm{I}}^{\dagger(i)}$ by sparse coding with $\mathbf{D}_{\mathrm{I}}^*$ $\forall i$
        Compute $\mathbf{z}_{\mathrm{I}}^{\dagger(i)}$ by max pooling $\forall i$
      else
        Compute $\mathbf{D}_{\mathrm{J}}^*$ by Eq. (9)
        Compute $\mathbf{y}_{\mathrm{J}}^{\dagger(i)}$ by sparse coding with $\mathbf{D}_{\mathrm{J}}^*$ $\forall i$
        Compute $\mathbf{z}_{\mathrm{J}}^{\dagger(i)}$ by max pooling $\forall i$
      end
    end
    Retrain classifier $h_{\mathbf{w}}$ with $\{\mathbf{z}_{L-2}^{\dagger(i)}, l^{(i)}\}$ $\forall i$
  end
until converged

IV. Experiments

We present a comparative performance analysis of single-layer sparse coding, deep stacked autoencoder (SAE) [13], and DSN on the multi-class image classification task using CIFAR-10. Both SAE and DSN have 4 layers.

A. Data processing and training for sparse coding

Instead of using the full CIFAR-10 dataset, we uniformly sample 20,000 images and cut to five folds for cross validation. We use four folds for training and the remaining fold for testing. We enforce exactly 2,000 images per class. For output layer, we have trained a 1-vs-all linear classifiers for each of ten classes in CIFAR-10. Each datum in CIFAR-10 is a 32×32×3 color image. We consider a *per-image* feature vector from densely overlapping patches drawn from a receptive field with width $w = 6$ pixels and stride $s = 2$. Thus, each patch (vectorized) has size $N = 3 \times 6 \times 6 = 108$. We preprocess patches by ZCA-whitening before sparse coding. We use a couple of different sparsity configurations for each LARS and OMP. We configure hidden layer 1 sparse coding more densely with $\lambda = 0.1$ (regularization penalty) for LARS and $S = 0.2N$ (sparsity bound) for OMP. For hidden layer 2, we use $\lambda = 0.2$ and $S = 0.1N$.

Figure 4 illustrates sparse coding and max pooling at hidden layer 1. Each image is divided into four quadrants. For each quadrant, there are four (pooling) groups of 9 patches. Hidden layer 1 uses a dictionary size $K_1 = 4N = 432$ and max pooling factor $M_1 = 9$. Hidden layer 1 produces $\{\mathbf{z}_{\mathrm{I}}^{(1)}, \ldots, \mathbf{z}_{\mathrm{I}}^{(4)}\}$, $\{\mathbf{z}_{\mathrm{I}}^{(5)}, \ldots, \mathbf{z}_{\mathrm{I}}^{(8)}\}$, $\{\mathbf{z}_{\mathrm{I}}^{(9)}, \ldots, \mathbf{z}_{\mathrm{I}}^{(12)}\}$, and $\{\mathbf{z}_{\mathrm{I}}^{(13)}, \ldots, \mathbf{z}_{\mathrm{I}}^{(16)}\}$ (4 pooled sparse codes per quadrant), which will be passed to hidden layer 2.

Figure 5 illustrates sparse coding and max pooling at hidden layer 2. We use $K_2 = 2K_1 = 864$ and $M_2 = 4$. To prevent further expansion of features, we have computed the two averages and encoded the final per-image feature vector $\phi_{DSN} = [\mathrm{mean}(\mathbf{z}_{\mathrm{II}}^{(1)}, \mathbf{z}_{\mathrm{II}}^{(2)}); \mathrm{mean}(\mathbf{z}_{\mathrm{II}}^{(3)}, \mathbf{z}_{\mathrm{II}}^{(4)})]$. Thus, the final feature vector for DSN has a dimensionality $2K_2 = 1,728$. For single-layer sparse coding, we have averaged $\mathbf{z}_{\mathrm{I}}^{(i)}$'s per quadrant and stacked them to form the final per-image feature vector ϕ_{SLSC}, which also has 1,728 features.

B. Data processing and training for autoencoder

For fairness, we have matched the total number of trained weights between the DSN and SAE schemes. Given an input $\mathbf{x} \in \mathbb{R}^N$, an autoencoder layer [1] trains a set of encoding weights that transforms $\mathbf{x}$ into the activations $\mathbf{a} \in \mathbb{R}^H$ and a set of decoding weights that recovers $\hat{\mathbf{x}}$, an estimate of $\mathbf{x}$, from $\mathbf{a}$, all through backpropagation. Similar to sparse coding, we use the encoding $\mathbf{a}$ as the feature vector for $\mathbf{x}$. If $H < N$, the autoencoder layer is forced to learn a *compressed* representation of the input. Also, even when $H > N$ (*i.e.*, more hidden units than input dimension), we can still learn meaningful features by imposing a *sparsity* constraint on $\mathbf{a}$ such that only $S \ll H$ neurons are activated. We have experimented with both approaches and will report the best result of the two for deep SAE.

TABLE I
AVERAGE 1-VS-ALL CLASSIFICATION ACCURACY FOR SINGLE-LAYER SPARSE CODING AND AUTOENCODER

	Classification accuracy
Autoencoder	69.8%
OMP ($S = 0.1N$)	75.3%
OMP ($S = 0.2N$)	76.9%
LARS ($\lambda = 0.2$)	78.4%
LARS ($\lambda = 0.1$)	**80.1**%

TABLE II
AVERAGE 1-VS-ALL CLASSIFICATION ACCURACY COMPARISON BETWEEN DSN AND SAE

	Classification accuracy
Deep SAE (pretraining only)	71.8%
Deep SAE (pretraining+backprop)	78.9%
DSN-OMP (pretraining only)	79.6%
DSN-OMP (pretraining+backprop)	84.3%
DSN-LARS (pretraining only)	83.1%
DSN-LARS (pretraining+backprop)	**87.5**%

C. Results

We report cross-validated 1-vs-all classification accuracy of single-layer sparse coding, DSN, and deep SAE. In Table I, we present the single-layer autoencoder and sparse coding performances. For classification, we have used two standard, off-the-shelf algorithms, SVM and logistic regression. The table summarizes the better of the two. LARS with $\lambda = 0.1$ has achieved the best single-layer accuracy at 80.1%.

In Table II, we compare the performance of DSN against deep SAE in various configurations. All DSN schemes show improvements from their respective single-layer configurations. Optimization by backpropagation over the whole network is critical for deep SAE as evidenced in the accuracy gain of 7.7%, which is significantly higher than the 2% gain from multilayering. For DSN, multilayering improves about 3%, and the proposed backpropagation additional 4%. Importantly, DSN-OMP with only pretraining is already 0.7% better than the backpropagation-finetuned deep SAE. DSN-OMP improves by 4.7% on backpropagation whereas the improvement is slightly less for DSN-LARS with a 4.4% gain. Overall, we find DSN-LARS the best performer.

V. CONCLUSION

Motivated by superior feature learning performance of single-layer sparse coding, we have presented Deep Sparse-coded Network (DSN), a deep architecture for sparse coding. We have discussed the benefit of DSN and described training methods including a novel backpropagation algorithm that effectively traverses and optimizes multiple layers of sparse coding and max pooling.

We stress that this paper limits to report an evaluation of DSN that confirms superior classification accuracy in a

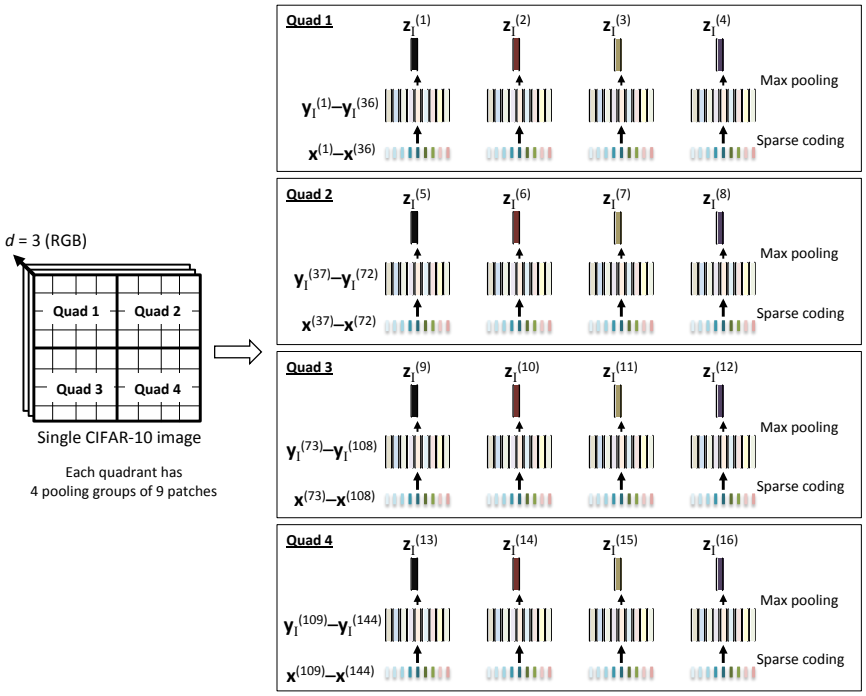

Fig. 4. Sparse coding and max pooling at hidden layer 1 for single CIFAR-10 image

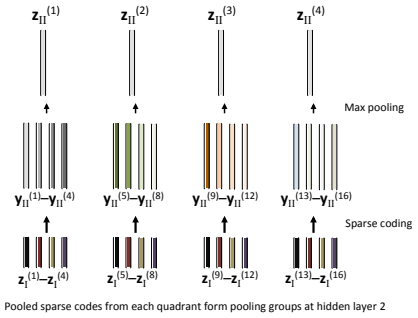

Fig. 5. Sparse coding and max pooling at hidden layer 2

medium-sized setup with CIFAR-10 images. We are currently experimenting with DSN more rigorously using larger datasets CIFAR-100, Caltech-101, and Caltech-256. In future work, we will test DSN in broader scope for text, sound, and wireless signal classification tasks.

ACKNOWLEDGMENTS

This work is supported in part by gifts from the Intel Corporation and in part by the Naval Supply Systems Command award under the Naval Postgraduate School Agreements No. N00244-15-0050 and No. N00244-16-1-0018.

REFERENCES

[1] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[2] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann Machines for Collaborative Filtering," in *ICML*, 2007.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," in *Proc. of the IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324.

[4] A. Coates, A. Y. Ng, and H. Lee, "An Analysis of Single-layer Networks in Unsupervised Feature Learning," in *AISTATS*, 2011.

[5] Y. Karklin and M. S. Lewicki, "Learning Higher-order Structures in Natural Images," *Network*, vol. 14, no. 3, pp. 483–99, 2003.

[6] C. Cadieu and B. A. Olshausen, "Learning Transformational Invariants from Natural Movies," in *NIPS*, 2008.

[7] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1994.

[8] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least Angle Regression," *Annals of Statistics*, vol. 32, pp. 407–499, 2004.

[9] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal Matching Pursuit: Recursive Function Approximation with Applications to Wavelet Decomposition," in *Asilomar Conference on Signals, Systems and Computers*, 1993.

[10] B. Olshausen and D. Field, "Sparse Coding with an Overcomplete Basis Set: Strategy Employed by V1?" *Vision research*, 1997.

[11] G. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[12] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation," *IEEE Trans. on Sig. Proc.*, vol. 54, no. 11, 2006.

[13] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.