
DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU


HARVARD LIBRARY
Office for Scholarly Communication

Deep Sparse-coded Network (DSN)

Citation
Gwon, Youngjune, Miriam Cha; H. T. Kung. 2016. Deep Sparse-coded Network (DSN). In Proceedings of 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, December 4-8, 2016. doi: 10.1109/ICPR.2016.7900029

Published Version
doi:10.1109/ICPR.2016.7900029

Permanent link
http://nrs.harvard.edu/urn-3:HUL.InstRepos:34903188

Terms of Use
This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Deep Spase-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 introduce Deep Sparse-coded Network (DSN), a deep architecture based on sparse coding and dictionary learning. Key advantage of our approach is two-fold. By interlacing max pooling with sparse coding layer, we achieve nonlinear activation analogous to neural networks, but suffering less from diminished gradients. We use a novel backpropagation algorithm to finetune our DSN beyond the pretraining by layer-by-layer sparse coding and dictionary learning. We build an experimental 4-layer DSN with the ℓ_1-regularized LARS and greedy-ℓ_0 OMP and demonstrate superior performance over deep stacked autoencoder on CIFAR-10.

1. 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 we cannot resolve ambiguity of the low-level (or localized) features of data. Another benefit of using deep architectures is representational efficiency. Deep architectures 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], sparse coding is found 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, it is much more than just stacking sparse coding units together. From sparse coding research on hierarchical feature learning [5,6], we could deduce plausible 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 indeed is quite cumbersome and challenging 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. This makes a straightforward adoption to take the output from one sparse coding unit for an input to another 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 present a deep architecture for sparse coding as a principled extension from its single-layer counterpart. We build on both the ℓ_1-regularized and greedy-ℓ_0 sparse coding. 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 can further optimize our performance.

Rest of this paper is organized as follows. In Section 2, we provide background on sparse coding. Section 3 will introduce Deep Sparse-coded Network (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.

2. SPARSE CODING BACKGROUND

Sparse coding is a general class of unsupervised methods to learn efficient representations of data as a linear combination of basis vectors in a dictionary. Given an input (patch) $\mathbf{x} \in \mathbb{R}^N$ drawn from the raw data and dictionary $\mathbf{D}^{N \times K}$, sparse coding searches for a representation (*i.e.*, sparse code) $\mathbf{y} \in \mathbb{R}^K$ in the ℓ_1-regularized optimization $\min_{\mathbf{y}} \|\mathbf{x} - \mathbf{D}\mathbf{y}\|_2^2 + \lambda\|\mathbf{y}\|_1$, known as LASSO or LARS. Greedy-ℓ_0 matching pursuit such as OMP is an alternative method for sparse coding in $\min_{\mathbf{y}} \|\mathbf{x} - \mathbf{D}\mathbf{y}\|_2^2$ s.t. $\|\mathbf{y}\|_0 \leq S$.

Dictionary learning is essential for sparse coding. During unsupervised feature learning, we perform sparse coding on unlabeled training examples by holding $\mathbf{D}$ constant. After sparse coding finishes, dictionary updates follow. They will alternate until convergence. Sparse coding can be thought as a generalization of K-means clustering that hard-assigns each training example to one cluster. Sparse coding can also be thought as less stringent, purely data-driven version of Gaussian mixture models.

3. DEEP SPARSE-CODED NETWORK (DSN)

3.1 Architectural Overview

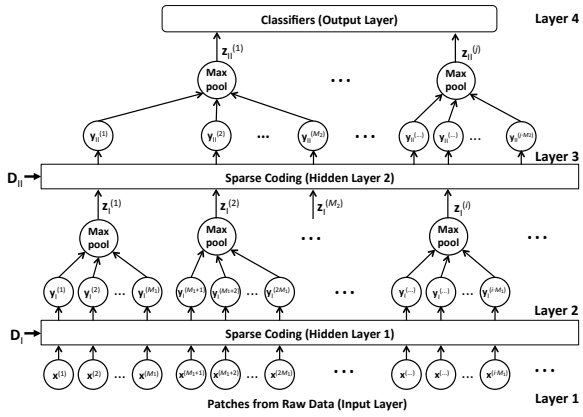

Figure 1: Deep Sparse-coded Network (DSN) with four layers

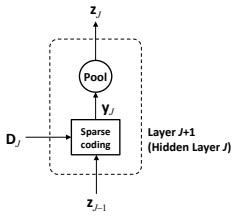

Figure 2: DSN layering module

Deep Sparse-coded Network (DSN) is a feedforward network built on multilayer sparse coding. We present a 4-layer DSN in Figure 1. This is a deep architecture because there are two hidden layers of sparse coding, each of which can learn corresponding level's feature representations by training own dictionary. 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 that are sparse coded and max pooled, propagating up the layers. Unlike convolutional neural network, we do not count pooling units as a separate layer. 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 $J \geq \text{II}$) 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 $J+1$.

3.2 Algorithms

Hinton *et al.* [7] suggested *pretrain* a deep architecture with layer-by-layer unsupervised learning and finetune via *backpropagation*, a supervised training algorithm popularized by neural network. We explain training algorithms for DSN using the example architecture in Figure 1.

3.2.1 Pretraining with layer-by-layer sparse coding and dictionary learning

DSN takes in *spatially* consecutive patches from an image or *temporally* consecutive patches from time-series data to make the overall feature learning meaningful. Optionally, 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 $\mathbf{y}_{\text{I}}^{(i)}$'s while learning $\mathbf{D}_{\text{I}}$ jointly

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

Max pooling at hidden layer 1 follows

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

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

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

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

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

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

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

3.2.2 Training classifiers at output layer

DSN learns each layer's dictionary greedily during pretraining. The resulting highest hidden layer output $\mathbf{z}_{\text{II}}$ 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 regression that takes a feature encoding ϕ as input. Note that ϕ is encoded on $\mathbf{z}_{\text{II}}$, but depends on DSN setup. For instance, we may have $\phi = [\mathbf{z}_{\text{II}}^{(1)}\ \mathbf{z}_{\text{II}}^{(2)}\ \mathbf{z}_{\text{II}}^{(3)}\ \mathbf{z}_{\text{II}}^{(4)}]^\top$ if the highest hidden layer yields four pooled sparse codes per training example.

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

3.2.3 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?

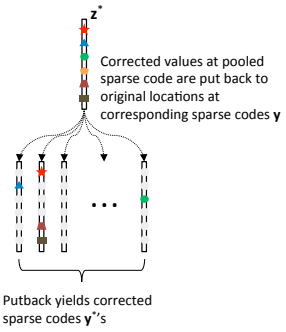

Figure 3: Putback corrects sparse codes y from $\mathbf{z}^*$

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

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

We define the loss or cost function for the DSN classification

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

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

$$z_{\mathrm{II},k} = z_{\mathrm{II},k} - \alpha\frac{\partial J(\mathbf{z}_\mathrm{II})}{\partial z_{\mathrm{II},k}} \tag{2}$$

where $\mathbf{z}_\mathrm{II} = [z_{\mathrm{II},1}\ z_{\mathrm{II},2}\ \ldots\ z_{\mathrm{II},K_2}]^\top$, 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 $z_{\mathrm{II},k}$

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

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

$$z_{\mathrm{II},k} := z_{\mathrm{II},k} + \alpha\left[l - h_\mathbf{w}(\mathbf{z}_\mathrm{II})\right]w_k \tag{3}$$

where α is the learning rate. 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 $\mathbf{y}^*_\mathrm{II}$'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 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 correct $\mathbf{D}_\mathrm{I}$, not $\mathbf{x}$, because it does not make sense to correct 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. The loss function with respect to $\mathbf{D}_\mathrm{I}$ is

$$J(\mathbf{D}_\mathrm{I}) = \frac{1}{2}\left\|\mathbf{D}_\mathrm{I}\mathbf{y}^*_\mathrm{I} - \mathbf{x}\right\|_2^2 \tag{4}$$

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

$$\min_{\mathbf{d}_{\mathrm{I},k}} J(\mathbf{D}_\mathrm{I})\ \text{ s.t. }\ \|\mathbf{d}_{\mathrm{I},k}\|_2^2 = 1\ \ \forall k \tag{5}$$

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

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

where $\mathbf{y}^*_\mathrm{I} = \left[y^*_{\mathrm{I},1}\ \ldots\ y^*_{\mathrm{I},K_1}\right]^\top$ and $\mathbf{y}_\mathrm{I} = [y_{\mathrm{I},1}\ \ldots\ y_{\mathrm{I},K_1}]^\top$. We obtain the update rule to adjust $\mathbf{D}_\mathrm{I}$ by gradient descent

$$\mathbf{d}_{\mathrm{I},k} := \mathbf{d}_{\mathrm{I},k} - \beta(\mathbf{D}_\mathrm{I}\mathbf{y}^*_\mathrm{I} - \mathbf{x})\left[y^*_{\mathrm{I},k} - y_{\mathrm{I},k}\right] \tag{6}$$

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

$$\mathbf{d}_{\mathrm{II},k} := \mathbf{d}_{\mathrm{II},k} - \gamma(\mathbf{D}_\mathrm{II}\mathbf{y}^*_\mathrm{II} - \mathbf{z}^\dagger_\mathrm{I})\left[y^*_{\mathrm{II},k} - y_{\mathrm{II},k}\right] \tag{7}$$

where $\mathbf{z}^\dagger_\mathrm{I}$ is the pooled sparse code over M_1 $\mathbf{y}^\dagger_\mathrm{I}$'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 $\mathbf{z}^\dagger_\mathrm{II}$ 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.

Algorithm 1 DSN backpropagation

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

repeat
  subalgorithm Down-propagation
    for J := $L-2$ **to** I
      if J == $L-2$
        Compute classifier error $\epsilon^{(i)} = l^{(i)} - h_\mathbf{w}(\mathbf{z}^{(i)}_{L-2})\ \ \forall i$
        Compute $\mathbf{z}^{*(i)}_{L-2}$ by $z^{(i)}_{L-2,k} = z^{(i)}_{L-2,k} + \alpha\cdot\epsilon^{(i)}\cdot w_k\ \ \forall i,k$
        Estimate $\mathbf{y}^{*(i)}_{L-2}$ from $\mathbf{z}^{*(i)}_{L-2}$ via putback $\ \forall i$
      else
        Compute $\mathbf{z}^{*(i)}_\mathrm{J} = \mathbf{D}_{\mathrm{J}+1}\mathbf{y}^{*(i)}_{\mathrm{J}+1}\ \ \forall i$
        Estimate $\mathbf{y}^{*(i)}_\mathrm{J}$ from $\mathbf{z}^{*(i)}_\mathrm{J}$ via putback $\ \forall i$
      end
    end
  end
  subalgorithm Up-propagation
    for J := I **to** $L-2$
      if J == I
        Compute $\mathbf{D}^*_\mathrm{I}$ by Eq. (6)
        Compute $\mathbf{y}^{\dagger(i)}_\mathrm{I}$ by sparse coding with $\mathbf{D}^*_\mathrm{I}\ \ \forall i$
        Compute $\mathbf{z}^{\dagger(i)}_\mathrm{I}$ by max pooling $\forall i$
      else
        Compute $\mathbf{D}^*_\mathrm{J}$ by Eq. (7)
        Compute $\mathbf{y}^{\dagger(i)}_\mathrm{J}$ by sparse coding with $\mathbf{D}^*_\mathrm{J}\ \ \forall i$
        Compute $\mathbf{z}^{\dagger(i)}_\mathrm{J}$ by max pooling $\forall i$
      end
    end
    Retrain classifier $h_\mathbf{w}$ with $\{\mathbf{z}^{\dagger(i)}_{L-2}, l^{(i)}\}\ \ \forall i$
  end
until converged

Table 1: Average 1-vs-all classification accuracy comparison

	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**%

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

$$\mathbf{z}_{\text{II}} \xrightarrow{\text{GD}} \mathbf{z}^*_{\text{II}} \xrightarrow{\text{putback}} \mathbf{y}^*_{\text{II}} \xrightarrow{\mathbf{D}_{\text{II}}} \mathbf{z}^*_{\text{I}} \xrightarrow{\text{putback}} \mathbf{y}^*_{\text{I}}$$

The corrections by up-propagation follow

$$\mathbf{D}_{\text{I}} \xrightarrow{\text{GD}} \mathbf{D}^*_{\text{I}} \xrightarrow{\text{SC}} \mathbf{y}^\dagger_{\text{I}} \xrightarrow{\text{max pool}} \mathbf{z}^\dagger_{\text{I}} \xrightarrow{\text{GD}} \mathbf{D}^*_{\text{II}} \xrightarrow{\text{SC}} \mathbf{y}^\dagger_{\text{II}} \xrightarrow{\text{max pool}} \mathbf{z}^\dagger_{\text{II}} \xrightarrow{\text{GD}} h_{\mathbf{w}}$$

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

4. EXPERIMENTAL RESULTS

We evaluate the classification performance of a 4-layer DSN using CIFAR-10.

4.1 Sparse coding setup

We denote different configurations of LARS and OMP by LARS-λ and OMP-ρ where $\rho = \frac{S}{K} \times 100$ (%). We configure hidden layer 1 sparse coding with more dense LARS-0.1 and OMP-20. For hidden layer 2, we use LARS-0.2 and OMP-10.

4.2 Data processing and training

Instead of using the full CIFAR-10 dataset, we uniformly sample 20,000 images and cut to four folds for cross validation. We use three folds for training and the remaining fold for testing. We have enforced the same number of 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 3×32×32 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. 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 $\{\mathbf{z}_{\text{I}}^{(1)}, \ldots, \mathbf{z}_{\text{I}}^{(4)}\}$, $\{\mathbf{z}_{\text{I}}^{(5)}, \ldots, \mathbf{z}_{\text{I}}^{(8)}\}$, $\{\mathbf{z}_{\text{I}}^{(9)}, \ldots, \mathbf{z}_{\text{I}}^{(12)}\}$, and $\{\mathbf{z}_{\text{I}}^{(13)}, \ldots, \mathbf{z}_{\text{I}}^{(16)}\}$ (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 $M_2 = 4$. We encoder the final per-image feature vector $\phi_{DSN} = [\mathbf{z}_{\text{II}}^{(1)}; \mathbf{z}_{\text{II}}^{(2)}; \mathbf{z}_{\text{II}}^{(3)}; \mathbf{z}_{\text{II}}^{(4)}]$. The final feature vector has a dimensionality $4K_2 = 3456$. (The feature vector is not too dense because of sparse coding.)

4.3 Results

We report cross-validated 1-vs-all classification accuracy of DSN against deep SAE. Both DSN approaches achieve better classification accuracy than deep SAE, and DSN-LARS is found to be the best performer. Optimization by backpropagation is critical for deep SAE as it gains more than 7% accuracy over pretraining only. DSN-OMP improves by 4.7% on backpropagation whereas the improvement is slightly less for DSN-LARS with a 4.4% gain. Importantly, DSN-OMP with pretraining only is already 0.7% better than deep SAE with both pretraining and backpropagation.

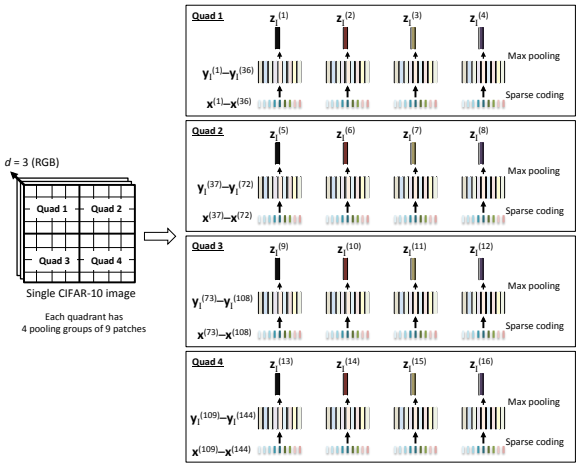

Figure 4: Sparse coding and max pooling at hidden layer 1 for single CIFAR-10 image.

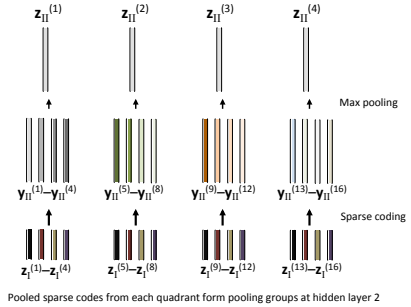

Figure 5: Sparse coding and max pooling at hidden layer 2 for single CIFAR-10 image.

5. CONCLUSION

Motivated by superior feature learning performance of single-layer sparse coding, we have presented Deep Sparse-coded Network (DSN), a deep architecture on multilayer sparse coding. DSN is a feedforward network having two or more hidden layers of sparse coding interlaced with max pooling units. We have discussed the benefit of DSN and described training methods including a novel backpropagation algorithm. From our experiments, we have found that DSN is superior to deep SAE in classifying CIFAR-10 images.

6. REFERENCES

[1] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[2] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *ICML*, 2007.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. In *Proc. of the IEEE*, volume 86, pages 2278–2324, 1998.

[4] Adam Coates, Andrew Y Ng, and Honglak Lee. An Analysis of Single-layer Networks in Unsupervised Feature Learning. In *AISTATS*, 2011.

[5] Yan Karklin and Michael S Lewicki. Learning Higher-order Structures in Natural Images. *Network*, 14(3):483–99, 2003.

[6] Charles Cadieu and Bruno A Olshausen. Learning Transformational Invariants from Natural Movies. In *NIPS*, 2008.

[7] G. Hinton, S. Osindero, and Y. Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 2006.