Doubly Sparsifying Network

Zhangyang Wang†, Shuai Huang‡, Jiayu Zhou§, Thomas S. Huang♭
†Department of Computer Science and Engineering, Texas A&M University
‡Department of Industrial and Systems Engineering, University of Washington
§ Department of Computer Science and Engineering, Michigan State University
♭ Beckman Institute, University of Illinois at Urbana-Champaign
atlaswang@tamu.edu, shuaih@uw.edu, jiayuz@msu.edu, t-huang1@illinois.edu

Abstract

We propose the *doubly sparsifying network* **(DSN)**, by drawing inspirations from the double sparsity model for dictionary learning. DSN emphasizes the joint utilization of both the problem structure and the parameter structure. It simultaneously sparsifies the output features and the learned model parameters, under one unified framework. DSN enjoys intuitive model interpretation, compact model size and low complexity. We compare DSN against a few carefully-designed baselines, and verify its consistently superior performance in a wide range of settings. Encouraged by its robustness to insufficient training data, we explore the applicability of DSN in brain signal processing that has been a challenging interdisciplinary area. DSN is evaluated for two mainstream tasks: electroencephalographic (EEG) signal classification and blood oxygenation level dependent (BOLD) response prediction, and achieves promising results in both cases.

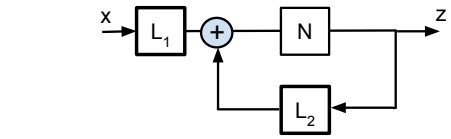
1 Introduction

With the prevailing success of deep models, it has been gradually recognized to incorporate the problem structure into the design of deep architectures. Such customized deep architectures can benefit from their problem-specific regularizations, and improve the performance as well as interpretability. In particular, there has been a blooming interest in bridging sparse coding [Elad and Aharon, 2006] and deep models. [Gregor and LeCun, 2010] first leveraged the idea to constructed feed-forward networks as fast trainable regressors to approximate the solutions of sparse coding models, which is followed by many recent works, e.g., [Sprechmann *et al.*, 2015], [Wang *et al.*, 2016a], [Wang *et al.*, 2016b]. Lately, [Xin *et al.*, 2016] demonstrated theoretically that a deep network could recover ℓ_0-based sparse representations under mild conditions.

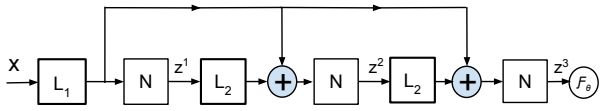
The paper proceeds along this direction to embed sparsity regularization into the target deep model, and simultaneously exploits the **structure of model parameters** into the design of the model architecture. Up to our best knowledge, it is the first unified framework that jointly sparsifies both learned features and model parameters. The resulting deep feed-forward network, called *doubly sparsifying network* **(DSN)**, enjoys a compact structure, a clear interpretation, an efficient implementation, and competitive performance, as verified by various comparison experiments. Its promising performance also manifests in the two novel application tasks of EEG signal classification and BOLD response prediction.

2 Related Work

2.1 Network Implementation of Sparse Coding



(a)



(b)

Figure 1: (a) The recursive system diagram for Eqn. (2); (b) a 3-layer neural network, unfolded and truncated to k = 2 iterations from (a).

We start from the classic sparse coding model [Elad and Aharon, 2006] ($||\mathbf{D}||_2$ = 1 by default hereinafter):

$$\mathbf{z} = \arg\min_{\mathbf{z}} \tfrac{1}{2}||\mathbf{x} - \mathbf{D}\mathbf{z}||_2^2 + \lambda||\mathbf{z}||_1. \quad (1)$$

$\mathbf{x} \in R^n$ denotes the input data, $\mathbf{z} \in R^m$ is the sparse code feature, $\mathbf{D} \in R^{n \times m}$ is the dictionary, and λ is the sparsity regularization coefficient. $\mathbf{D}$ is usually chosen to be *overcomplete*, i.e. $m > n$. Eqn. (1) can be solved by the iterative shrinkage and thresholding algorithm (ISTA) [Blumensath and Davies, 2008] ($\mathbf{u}$ is a vector and u_i is its i-th element):

$$\begin{aligned} &\mathbf{z}^{k+1} = \mathcal{N}(\mathcal{L}_1(\mathbf{x}) + \mathcal{L}_2(\mathbf{z}^k)), \text{ where :} \\ &\mathcal{L}_1(\mathbf{x}) = \mathbf{D}^T\mathbf{x},\ \mathcal{L}_2(\mathbf{z}^k) = (\mathbf{I} - \mathbf{D}^T\mathbf{D})\mathbf{z}^k, \\ &\mathcal{N}(\mathbf{u})_i = \text{sign}(u_i)(|u_i| - \lambda)_+, \end{aligned} \quad (2)$$

where $\mathbf{z}^k \in R^m$ denotes the intermediate output of the k-th iteration, k = 0, 1, $\cdots$. $\mathcal{L}_1$ and $\mathcal{L}_2$ are linear operators that both hinge on $\mathbf{D}$, while $\mathcal{N}$ is the element-wise soft shrinkage.

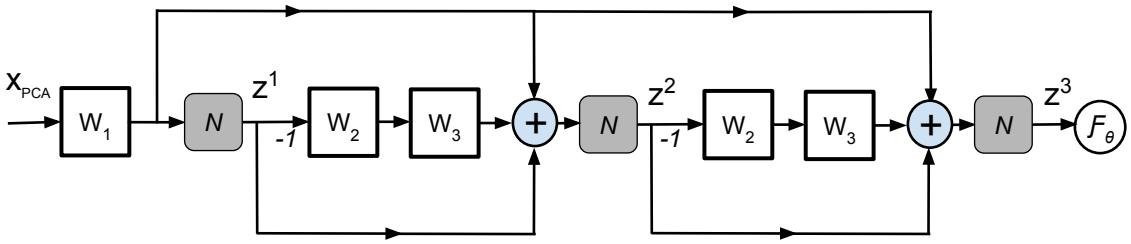


Figure 2: The proposed doubly sparsifying network, unfolded and truncated to k = 2 iterations. The parameters $\mathbf{W}_l$ (l = 1, 2, 3) are subject to the constraints in Eqn. (6).

Eqn. (2) is equivalently expressed by the recursive system in Figure 1 (a), whose fixed point is expected to be the solution of (1). Moreover, Figure 1 (a) could be *unfolded* and *truncated* to k iterations, to construct a (k+1)-layer feed-forward network [Gregor and LeCun, 2010], as in Figure 1 (b). Without any further tuning, the resulting *learned ISTA* (LISTA) architecture will output a k-iteration approximation of the exact solution $\mathbf{a}$. Figure 1 (b) could be further viewed as a *trainable regressor* to fit the data, as a function of $\mathbf{D}$. It could be jointly tuned with a task-specific loss function $\mathcal{F}_\theta(\mathbf{z}^k)$ (e.g., the softmax loss for classification; θ denotes the parameters of the loss function), as an end-to-end network [Wang *et al.*, 2016d].

2.2 Double Sparsity Model for Dictionary Learning

A crucial consideration in employing the sparse coding model (1) is the choice of the dictionary $\mathbf{D}$. It has been observed that for structured signals (e.g., image, speech), the learned dictionary is also highly structured, with noticeably regular patterns. This gives rise to the hypothesis that the dictionary atoms themselves may have underlying sparse structures over a more fundamental dictionary. [Rubinstein *et al.*, 2010] proposed a double sparsity model, suggesting that each dictionary atom has a sparse representation over some pre-specified base dictionary $\mathbf{D}_0$, expressed as:

$$\mathbf{D} = \mathbf{D}_0\mathbf{S}, ||\mathbf{S}(:,i)||_0 \leq s, \forall i, \tag{3}$$

where $\mathbf{S}$ is the sparse atom representation matrix, which has no more than s nonzero elements per column ($s \ll n, m$). We also assume $\mathbf{D}_0 \in R^{n\times n}$ and $\mathbf{S} \in R^{n\times m}$. Note that in [Rubinstein *et al.*, 2010], $\mathbf{D}_0$ is chosen as $R^{n\times m}$, and $\mathbf{S} \in R^{m\times m}$. We make slightly different choices to ensure that $\mathbf{D}_0$ is unitary, the reason of which will be seen next. The base dictionary $\mathbf{D}_0$ spans the signal space, and is generally chosen to have an efficient implementation. The new parametric structure of $\mathbf{D}$ leads to an adaptive and efficient dictionary representation. Advantages of the double sparsity model (3) also include compact representation, stability under noise and reduced overfitting, among others.

3 Doubly Sparsifying Network

3.1 The Proposed Model

Given $\mathbf{D}_0$ and $\mathbf{S}$, we substitute (3) into (2) to obtain:

$$\mathcal{L}_1(\mathbf{x}) = \mathbf{S}^T\mathbf{D}_0^T\mathbf{x},\ \mathcal{L}_2(\mathbf{z}^k) = (\mathbf{I} - \mathbf{S}^T\mathbf{D}_0^T\mathbf{D}_0\mathbf{S})\mathbf{z}^k, \tag{4}$$

with the iterative formula of $\mathbf{z}^k$ and the form of $\mathcal{N}$ remaining the same. Compared to (2), $\mathbf{S}$ now becomes the trainable parameter in place of $\mathbf{D}$.

To simplify (4), we first eliminate $\mathbf{D}_0^T\mathbf{D}_0$ from $\mathcal{L}_2(\mathbf{z}^k)$. Given the training data $\mathbf{X}_\Sigma \in R^{n\times t} = \{\mathbf{x}_i\}, i = 1, 2, ..., t$, and assuming $\mathbf{X}_\Sigma$ to have zero mean, we choose $\mathbf{D}_0$ as the (full) eigenvector matrix of $\mathbf{X}_\Sigma\mathbf{X}_\Sigma^T$ (i.e., the covariance matrix of $\mathbf{X}_\Sigma$). The obtained $\mathbf{D}_0$ constitutes an orthonormal basis for R^n. Further, $\mathbf{D}_0^T\mathbf{x}$ performs the PCA projection of $\mathbf{x}$, denoted as: $\mathbf{x}_{\text{PCA}} = \mathbf{D}_0^T\mathbf{x}$. The formula (4) is reduced to:

$$\begin{aligned}&\mathcal{L}_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x}_{\text{PCA}},\ \mathcal{L}_2(\mathbf{z}^k) = (\mathbf{I} - \mathbf{W}_3\mathbf{W}_2)\mathbf{z}^k, \text{ where}\\ &\mathbf{W}_1 = \mathbf{S}^T, \mathbf{W}_2 = \mathbf{S}, \mathbf{W}_3 = \mathbf{S}^T.\end{aligned} \tag{5}$$

We introduce three new variables in (5): $\mathbf{W}_1 \in R^{m\times n}$, $\mathbf{W}_2 \in R^{n\times m}$, and $\mathbf{W}_3 \in R^{m\times n}$. Both $\mathbf{W}_1$ and $\mathbf{W}_3$ have no more than s nonzero elements per *row*, while $\mathbf{W}_2$ has no more than s nonzero elements per *column*. Figure 2 depicts the resulting *doubly sparsifying network* **(DSN)**, unfolded and truncated from (5) (up to k = 2). We purposely model $\mathbf{W}_2$ and $\mathbf{W}_3$ as two separate layers (with no nonlinearity in between), so that we could specify the proper row- or column-wise sparsity constraint on each. That is similar to constructing one linear layer, which is required to have a special sparse matrix factorization (SMF) form [Neyshabur and Panigrahy, 2013].

Furthermore, under the loss function $\mathcal{F}_\theta$, $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_3$ can again be learned via end-to-end learning, instead of being constructed from any pre-computed $\mathbf{S}$[1]. In this way, the DSN network is solved over $\mathbf{D}_0^T\mathbf{X}_\Sigma$ by back-prorogation, where $\mathbf{W}_l$ (l = 1, 2, 3) are treated as fully-connected layers. Different from [Wang *et al.*, 2016b], we find it helpful to untie $\mathbf{W}_2$ and $\mathbf{W}_3$ throughout iterations, in order for larger learning capacity. During training, we also relax the formulation (5), by decoupling $\mathbf{W}_l$ (l = 1, 2, 3) with each other, e.g., it is no longer required that $\mathbf{W}_1$ = $\mathbf{W}_3$, or $\mathbf{W}_2^T$ = $\mathbf{W}_3$. For simplicity, we use the same s for all $\mathbf{W}_l$s.

3.2 The Projected Gradient Descent Algorithm

Let $\mathcal{G}$ denote the nonlinear mapping from the data to the last hidden feature before the loss, the optimization problem of training DSN could be written as below:

$$\begin{aligned}&\min_{\{\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3,\theta\}} \mathcal{F}_\theta(\mathcal{G}(\mathbf{X}_\Sigma|\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3)),\\ &s.t. ||\mathbf{W}_1(i,:)||_0 \leq s, ||\mathbf{W}_2(:,j)||_0 \leq s,\\ &\quad ||\mathbf{W}_3(k,:)||_0 \leq s, \forall i,j,k.\end{aligned} \tag{6}$$

[1] Another parameter to be learned jointly is the threshold λ in $\mathcal{N}$. It is handled identically as in [Wang *et al.*, 2016c].

Apart from the constraints, the objective in (6) is usually minimized by the (stochastic) gradient descent (SGD) algorithm:

$$\mathbf{W}_l = \mathbf{W}_l - \gamma \frac{\partial \mathcal{F}}{\partial \mathbf{W}_l}, l = 1, 2, 3, \quad (7)$$

where γ is the learning rate. With the constraints in (6) specifying the feasible sets, we naturally obtain the Projected Gradient Descent (PGD) algorithm:

$$\mathbf{W}_l = \mathcal{P}_l(\mathbf{W}_l - \gamma \frac{\partial \mathcal{F}}{\partial \mathbf{W}_l}), l = 1, 2, 3. \quad (8)$$

where $\mathcal{P}_l$ is the projection onto the feasible set for $\mathbf{W}_l$ [Blumensath and Davies, 2008]. When l = 1, 3, $\mathcal{P}_l$ keeps the s largest-magnitude elements in each row of $\mathbf{W}_l$, and zeros out others. For l = 2, $\mathcal{P}_l$ is the same hard thresholding operator, but on a column-wise basis.

SGD is guaranteed to converge to a stationary point, under a few stricter assumptions than the ones satisfied here [Bottou, 2010]. Since both the objective and feasible sets of (6) are non-convex, there is no convergence guarantee for PGD in (8). However, many literatures, such as [Blumensath and Davies, 2008], have demonstrated that solving such problems with PGD is well executed in practice. The stochastic implementation of PGD is also straightforward.

3.3 Complexity Analysis

Model parameter complexity

For k-iteration DSN, each $\mathbf{W}_l$ (l= 1, 2, 3) is a sparse matrix of sm nonzero elements. The total amount of parameters in DSN is $(2k + 1)sm$. In contrast, the LISTA network in Figure 1 (b) takes $mn + km^2$ parameters, assuming its $\mathcal{L}_2$ parameters not tied across iterations as well. Since $s \ll m, n$, the parameter ratio turns out to be $\frac{(2k+1)sm}{mn+km^2} = \frac{(2k+1)s}{n+km} \to \frac{2s}{m} \ll 1$, as $k \to \infty$. DSN can thus be stored and loaded much more compactly, due to the sparse structure of $\mathbf{W}_l$s. More importantly, **DSN can ensure the sufficient capacity and flexibility of learning by using large m, meanwhile regularizing the learning process by choosing small s.**

Inference time complexity

The efficient multiplication of a sparse matrix with sm nonzero elements, and an arbitrary input vector, takes sm time. Given a k-iteration DSN, the inference time complexity of one sample $\in R^n$ is $\mathcal{O}((2k + 1)sm)$. In comparison, LISTA has a time complexity of $\mathcal{O}(mn + km^2)$. Again, when $k \to \infty$, $\frac{(2k+1)sm}{mn+km^2} \to \frac{2s}{m} \ll 1$.

Remark on the number of layers

When (5) is unfolded and truncated to k iterations, the obtained DSN has 1 $\mathbf{W}_1$ layer, k $\mathbf{W}_2$ layers, and k $\mathbf{W}_3$ layers. However, since $\mathbf{W}_2$ and $\mathbf{W}_3$ are always linearly concatenated within each iteration, with no nonlinearity in between, we can also consider $\mathbf{W}_3\mathbf{W}_2 \in R^{m\times m}$ as one layer, whose two factors are individually regularized. Hence, we treat a DSN unfolded to k iterations as a (k+1)-layer network, which also follows the LISTA convention [Gregor and LeCun, 2010].

3.4 Relationship to Existing Works

Many regularization techniques have been proposed to reduce overfitting in deep learning, such as dropout [Krizhevsky *et al.*, 2012], that set a randomly selected subset of activations to zero within each hidden layer. [Wan *et al.*, 2013] introduced *dropconnect* for regularizing fully-connected layers, which instead sets a randomly selected subset of weights to zero during training. In comparison, DSN reveals an adaptive regime for dropconnect, where the selection of "dropped" weights is decided not randomly, but by data-driven hard thresholding. Besides, both dropout and dropconnect are only applied to training, and are unable to reduce the actual model size. DSN could also be viewed to have a *weight decay* penalty, which is enforced by hard ℓ_0 constraints.

A recent work [Jin *et al.*, 2016] also imposed explicit layer-wise cardinality constraints to the network parameters during training. However, DSN more clearly interprets the motivation to sparsify parameters, by referring to the inherent sparse structure of dictionary resulting from the structured signal space. Moreover, [Jin *et al.*, 2016] focused more on the general training strategy, while DSN combines pursuing sparse features with pruning model parameters. It is thus potentially more favored by many discriminative feature learning tasks where feature sparsity is explicitly desirable [Coates and Ng, 2011].

4 Simulation and Analysis

4.1 Implementation

The proposed DSN is implemented with the CUDA ConvNet package [Krizhevsky *et al.*, 2012]. We use a constant learning rate of 0.01, with the momentum parameter fixed at 0.9, and a batch size of 128. Neither dropout nor dropconnect is applied unless specified otherwise. We manually decrease the learning rate when the validation error of the network stops decreasing.

As suggested by (5), we first subtract the mean and conduct PCA over the training data $\mathbf{X}_\Sigma$. We adopt the multi-step update strategy in [Jin *et al.*, 2016], namely, updating $\mathbf{W}_l$ by SGD without the cardinality constraints for several (15 by default) iterations, before the projection $\mathcal{P}_l$ (l = 1, 2, 3). It both accelerates training by reducing the time of performing hard thresholding, and forces DSN to learn more informative parameters to make pruning more reliable.

While many neural networks are trained well with random initializations, it has been discovered that poor initializations can still hamper the effectiveness of first-order methods [Sutskever *et al.*, 2013]. On the other hand, It is much easier to initialize DSN in the right regime. We first initialize $\mathbf{S}$ by setting s randomly selected elements to be one for each column, and zero elsewhere: that ensures a random starting point within the feasible space, as specified by the constraints. Based on the correspondence relationships in (5), $\mathbf{W}_l$s (l= 1, 2, 3) are all trivially initialized from $\mathbf{S}$. That helps DSN achieve a more steadily decreasing curve of training errors.

4.2 Comparison

In the simulation experiments, we use the first 60, 000 samples of the MNIST dataset for training and the last 10,000 for testing. Each MNIST sample is a 28×28 gray-scale image, i.e., n = 784. Common data augmentations (noice, blur, rotation, and scaling) are applied. In addition to a k-iteration DSN, we design five baselines for comparison:

- **Baseline I:** (k+1)-layer fully-connected network, whose first layer $\in R^{m\times n}$ and remaining k layers $\in R^{m\times m}$.
- **Baseline II:** Baseline I regularized by *dropout*, with a ratio of 0.5 for each layer [Krizhevsky *et al.*, 2012].
- **Baseline III:** Baseline I regularized by *dropconnect*, with a ratio of 0.5 (as in [Wan *et al.*, 2013]) for each layer.
- **Baseline IV:** a LISTA network, unfolded and truncated to k iterations from (1). We also apply dropout to regularize its fully-connected layers.
- **Baseline V:** a network inspired by [Jin *et al.*, 2016], by removing all "shortcuts" in DSN.

All comparison methods are ensured to have the identical layer dimensions. They are jointly tuned with the softmax loss for the classification task. The default configuration parameters are $s = \frac{1}{4}n$, $m = 1,024$, $t = 60,000$, and $k = 2$. We further vary each of the four parameters, while keeping others unchanged, in our controlled experiments below.

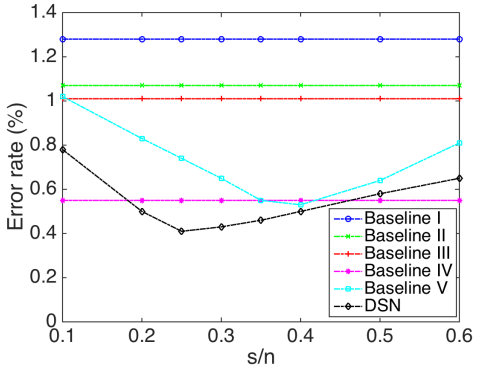


Figure 3: The error rate (%) comparison between baselines and DSN on MNIST, with the sparsity ratio s/n varied.

Sparsity level s

Figure 3 varies the sparsity ratio s/n from 0.1 to 0.6, and plots the corresponding error rates for all methods. Baselines I - IV are not parameterized by s and thus not affected. Comparing Baselines II and III with Baseline I certifies that applying (even random) regularizations avoids overfitting and improves generalization. Baseline V and DSN both benefit further from their more sophisticated regularization on the parameters. DSN outperforms Baseline V with noticeable margins at all s/n ratios, and reaches the best overall performance at $s/n = 0.25$.

As displayed in Figure 3, the performance of Baseline V and DSN will both be degraded with either too small or too large s/n ratios. Whereas increasing s/n may loose the regularization effect, a small s/n also implies over-regularization, limiting the representation power of free parameters. In the random dropout/dropconnect cases, the popular practice is to choose s/n around 0.5. [Jin *et al.*, 2016] also observed the best s/n to be between 0.4 and 0.5. DSN seems to admit a lower "optimal" s/n (around 0.25). It implies that DSN could attain more competitive performance with fewer parameters (i.e., lower s/n), by "smartly" selecting non-zero elements in a data-driven way.

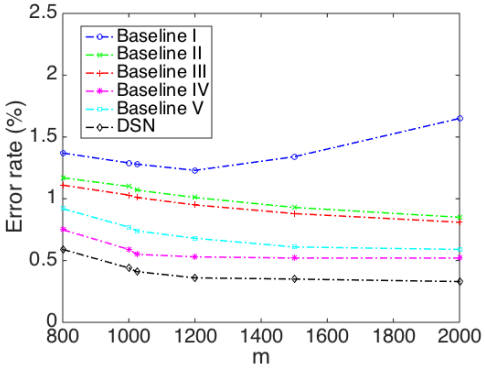


Figure 4: The error rate (%) comparison between baselines and DSN on MNIST, with the feature dimension m varied.

Feature dimension m

In (1), the choice of m corresponds to the dimensionality of the learned sparse code feature, and turns into the hidden layer dimensions of DSN, etc. As illustrated in Figure 4, we start from $m = 800$, and raise it up to 2, 000. Not surprisingly, the performance of Baseline I is degraded with m growing larger, due to obviously overfitting. All other methods, regularized in various ways, all seem to benefit from larger m values. Among them, DSN consistently outperforms others, with a 0.2% error rate margin over Baseline IV (the second best). It proves effective to handle highly over-complete and redundant basis, and hence to learn more sparse hidden features.

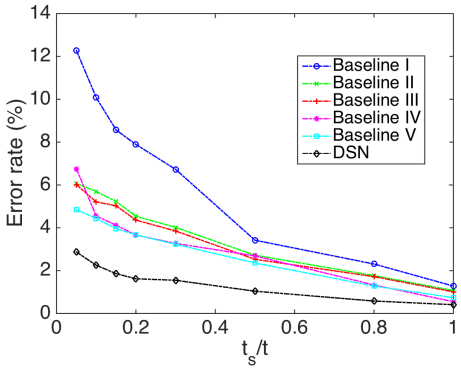


Figure 5: The error rate (%) comparison between baselines and DSN on MNIST, with t_s/t varied.

Training sample size $t(t_s)$

DSN seeks a trade-off between "data-driven" and "model-based" methods. By confining the degrees of freedom of parameters and permitting only certain sparse combinations over a pre-specified base dictionary, the parameter structure model (3) enables us to reduce, in some cases significantly, the amount of training data required to reliably approximate and recover the underlying nonlinear mapping of the deep model.

We empirically verify our conjecture, by the following comparison experiment. A small subset of size t_s is drawn from $\mathbf{X}_\Sigma$ (the MNIST dataset with $t = 60,000$ samples), where

each class is sampled proportionally. We range the ratio t_s/t from 0.1 to 1. Figure 5 shows that DSN leads to dramatically more robust learning and generalization, under insufficient training data. Even when t_s/t is as low as 0.05, DSN only bears a slight performance loss of 2.46%, while Baselines IV and V are degraded for more than 6% and 4%, respectively. It is also noteworthy that, to achieve the same performance level of DSN at $t_s/t = 0.05$, Baselines IV and V requires approximately $t_s/t = 0.4$, Baselines II and III take $t_s/t = 0.5$, and Baseline I even needs $t_s/t \geq 0.8$. Those observations strongly support our hypothesis, that DSN greatly alleviates the need for large training data, by exploiting the prior knowledge of parameter structure. In addition, we note that Baseline V slightly outperforms Baseline IV in Figure 5. Recall that similarly to DSN, the regularization on the Baseline V parameters is also enforced by the data-driven adaptive sparsity. Under small training data, it is shown more effective than the "random sparsity" induced by dropout.

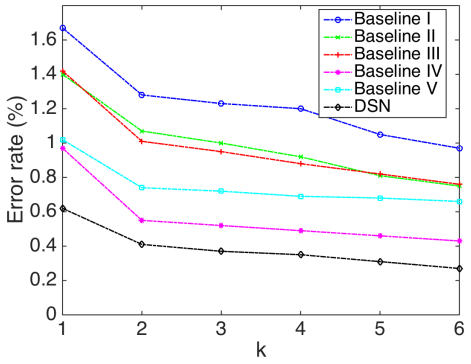


Figure 6: The error rate (%) comparison between baselines and DSN on MNIST, with k varied.

Number of layers k + 1

The last thing that we investigate is how well DSN and other methods can scale to deeper cases. We grow k from 1 to 6, resulting in 2 to 7-layer networks[2]. The comparison in Figure 6 evidently demonstrates the superiority of DSN at all k values. Besides, it is also interesting to see from Figure 6, that Baseline IV obtains a significant performance advantage over Baseline V as k grows. It is opposite to the observation in Figure 5. On one hand, it might be attributed to the utility of "shortcuts", as analyzed in [He *et al.*, 2016]. On the other hand, we believe that the incorporation of the original problem structure (1) also places deep models in good conditions: increasing k is resemblant to running (2) up to more iterations, and thus solving (1) more precisely.

Remarks

We conclude from the above experiments, that both the problem structure ("sparsifying features") and the parameter structure ("sparsifying parameters') have contributed to the superior performance of DSN.

[2]We apply layer-wise pre-training [Erhan *et al.*, 2010] to Baseline I when $k > 2$, to ensure that it converges properly.

By the comparison to Baselines II and III, the sophisticated regularization of DSN is found to be more powerful than random ones such as dropout/dropconnect. Compared to Baseline IV, DSN further utilizes the double sparsity structure of the dictionary (3) as a priori, which accounts for its improved performance in all aspects. In the meanwhile, exploiting the structure of the original problem (1), that encourages sparse and more discriminative features, also helps DSN outperform Baseline V consistently.

Besides, although the simulations are only intended for proof-of-concepts, the result of default-configured DSN has already had comparable results to the 6-layer neural network in [Ciresan *et al.*, 2010], and the committee of 25 neural networks trained with elastic distortions in [Meier *et al.*, 2011].

5 Applications in Brain Signal Processing

DSN shows superior performance in simulations. In particular, the experiments on varying $t(t_s)$ identified a sharp performance margin for DSN over the others. It thus implies the possibility for DSN to handle smaller data. In this paper, we explore the applicability of DSN in brain signal processing, a challenging interdisciplinary area that calls for computational innovations. EEG signal classification and BOLD response prediction have been two of its mainstream tasks that particularly fit the concept of DSN.

We are motivated to apply DSN for brain signal processing, primarily because the sample sizes are very limited in most existing brain signal datasets, with only tens of participants. Unlike common computer vision tasks [Krizhevsky *et al.*, 2012], it is usually infeasible to perform artificial data augmentation or generate "synthetic" data here. Therefore, models that are economic in free parameters are advantageous. The sparsely-structured parameters of DSN show promise in saving parameters and avoiding overfitting, without hampering model effectiveness. Moreover, brain signals are known to be highly structured. The sensory processing in the brain suggests a sparse coding strategy over an over-complete basis [Olshausen and Field, 1997]. Such a neuroscience ground encourages us to sparsify the features in the brain encoding models. As the ultimate goal to study brain signal processing is to understand how brain works, it is interesting to verify the underlying low-dimensional structure of brain signals from a learning perspective too.

5.1 EEG Signal Classification

Brain-computer interfaces (BCI) enable the control of the external environment through direct measures of brain signals [Wolpaw *et al.*, 2000]. Its applications can be found in a broad range of fields such as rehabilitation engineering, military, and entertainment. Electroencephalographic (EEG) signal classification belongs to the core tasks of BCI. Previous literatures explored band power [Brodu *et al.*, 2011], multivariate adaptive autoregressive (MVAAR) [Anderson *et al.*, 1998], and independent component analysis (ICA) [Hung *et al.*, 2005], among many others. Recent works [Ren and Wu, 2014; Tabar and Halici, 2016] showed that that deep networks yield superior results on this task.

We follow [Tabar and Halici, 2016] to adopt the benchmark dataset 2b from BCI Competition IV (training set) [Schlögl,

Table 1: The accuracy (%) comparison on BCI Competition IV dataset 2b among SVM, CNN, CNN-SAE and DSN.

Subject	SVM	CNN	CNN-SAE	DSN
1	71.8	74.5	**76.0**	74.8
2	64.5	64.3	65.8	**69.6**
3	69.3	71.8	**75.3**	74.3
4	93.0	94.5	95.3	**97.5**
5	77.5	79.5	83.0	**87.5**
6	72.5	75.0	79.5	**82.0**
7	68.0	70.5	74.5	**77.0**
8	69.8	71.8	75.3	**77.3**
9	65.0	71.0	**75.3**	67.6
Average	72.4	74.8	77.6	**78.7**

2003]. The dataset includes three sessions of motor imagery task experiments, with two classes of motions: right and left hand movements. The EEG signal classifier thus solves a binary classification problem: to distinguish EEG signals associated with right hand movements from those associated with left hand movements. We train and test models separately for each of nine subjects. In each session, we randomly select 90% of 400 trials for training and the remaining 10% for testing, and report the mean accuracy of 10 runs.

We transform the extracted signal to construct the input using the same protocol as [Tabar and Halici, 2016]. Specifically, they introduced to pre-process the input, by pre-training a 1-D convolutional neural network (CNN) with one convolutional layer and one max-pooling layer. It was shown to preserve the input spatial pattern better than the straightforward option of vectorization and PCA. The pre-processed output was then classified by a 7-layer stacked auto-encoder (SAE). Besides the proposed *CNN-SAE* method, the authors also reported two baselines: applying only the CNN part for classification, and directly classifying the input by SVM. We follow their CNN pre-processing strategy, and use a 1-layer CNN with 10 channels of 1-D convolutional filter of size 20. The input n is thus 200. Other configuration hyperparameters are chosen as $k = 3$, $m = 256$, and $s = 64$. Note that DSN costs fewer parameters than the CNN-SAE architecture used in [Tabar and Halici, 2016]. Table 1 clearly shows the performance advantage of DSN over the competitive CNN-SAE method: DSN outperforms in six out of nine subjects, and maintains a 1% margin in terms of the average accuracy.

5.2 BOLD Response Prediction

Brain encoding has been held as a crucial technical step to advance many areas such as brain-computer interaction, rehabilitation, and situational awareness enhancement for military applications. It essentially concerns the prediction of brain activity, e.g, the blood oxygenation level dependent (BOLD) response, with a given stimuli. For example, with a grayscale image as the stimuli, [Kay *et al.*, 2013] developed a two-stage cascaded *second-order contrast* (**SOC**) model to predict the BOLD responses in early visual cortex. The SOC model has only eight controlling parameters: it heavily relies on specific nonlinear computations, that are summarized from neuroscience expertise. While many existing models such as [Kay *et al.*, 2013] used highly domain-knowledge driven models, we propose to study DSN as a more parameterized and flexible option, that could lead to the data-driven discovery of unknown structures embedded in the brain signals.

We refer to Kendrick Kay et.al.'s publicly available datasets of BOLD responses in visual cortex[3], measured by functional magnetic resonance imaging (fMRI) in human subjects. We use their stimulus set 2, stimulus set 3, (response) dataset 4, and (response) dataset 5. All stimuli are band-pass filtered grayscale synthetic images. Following [Kay *et al.*, 2013], we resize all the stimuli to 150×150 pixels. Stimulus sets 2 and 3 consist of **156** and **35** distinct stimuli, respectively. The responses at a total of 200 voxels are recorded. On each voxel, a scalar fMRI measurement was measured given each input stimuli. Note that each voxel needs to train a separate brain encoding model. Dataset 4 consists of one person's responses to stimulus set 2, while dataset 5 has the same person's responses to stimulus set 3. Details about the datasets could be found at [Kay *et al.*, 2013].

Our goal is to train a regression-type model using stimulus set 2 and dataset 4 (as the *training set*). The model is used to generate predictions and be evaluated on stimulus set 3 and dataset 5 (as the *testing set*). Obviously, it is a highly ill-conditioned "small data" problem. To adapt DSN for the regression task, we replace the softmax function, with a max pooling operator: $R^m \to R$, followed by a mean square error (MSE) loss. Due to the very simple nature of the used visual stimuli, we vectorize and project each image into a 128-d vector for input, and choose $k = 2$, $m = 256$, and $s = 64$. A 3-layer fully-connected (FC) network is also constructed for comparison, with three hidden layers of 256-dimension, and regularized by dropout with a ratio of 0.5.

Our experiment has found encouraging results. The model accuracy is quantified as the percentage of variance explained (R^2) in the measured response amplitudes by the cross-validated predictions of the response amplitudes. Following [Kay *et al.*, 2013], R^2 is defined to range between [0, 100]: the higher R^2 is, the more accurate the model is. In terms of *averaged* R^2 performance across 200 voxels, the SOC baseline obtains 87.7028, and the averaged R^2 for FC is as poor as 69.5077. DSN leads to the best averaged R^2 of 88.4026. In future, it is natural to extend (1) and (3) to the convolutional counterparts, and to apply them to the BOLD response prediction of visual stimuli.

6 Summary

The study of DSN showcases how jointly exploiting the problem structure and the parameter structure improves the deep modeling. Both simulations and the two brain signal processing applications have verified its consistently superior performance, as well as robustness to highly limited training data. In our future work, a wide variety of parameter structures will be exploited for different models as a priori.

References

[Anderson *et al.*, 1998] Charles W Anderson, Erik A Stolz, and Sanyogita Shamsunder. Multivariate autoregressive

[3] http://kendrickkay.net/socmodel/

models for classification of spontaneous electroencephalographic signals during mental tasks. *IEEE Transactions on Biomedical Engineering*, 45(3):277–286, 1998.

[Blumensath and Davies, 2008] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 2008.

[Bottou, 2010] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010.

[Brodu *et al.*, 2011] Nicolas Brodu, Fabien Lotte, and Anatole Lécuyer. Comparative study of band-power extraction techniques for motor imagery classification. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*, 2011.

[Ciresan *et al.*, 2010] DC Ciresan, U Meier, LM Gambardella, and J Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. 2010.

[Coates and Ng, 2011] Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, 2011.

[Elad and Aharon, 2006] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE TIP*, 2006.

[Erhan *et al.*, 2010] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *JMLR*, 2010.

[Gregor and LeCun, 2010] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE CVPR*, 2016.

[Hung *et al.*, 2005] Chih-I Hung, Po-Lei Lee, Yu-Te Wu, Li-Fen Chen, Tzu-Chen Yeh, and Jen-Chuen Hsieh. Recognition of motor imagery electroencephalography using independent component analysis and machine classifiers. *Annals of Biomedical Engineering*, 33(8):1053–1070, 2005.

[Jin *et al.*, 2016] Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

[Kay *et al.*, 2013] Kendrick N Kay, Jonathan Winawer, Ariel Rokem, Aviv Mezer, and Brian A Wandell. A two-stage cascade model of bold responses in human visual cortex. *PLoS Comput Biol*, 9(5):e1003079, 2013.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[Meier *et al.*, 2011] Ueli Meier, Dan Claudiu Ciresan, Luca Maria Gambardella, and Jurgen Schmidhuber. Better digit recognition with a committee of simple neural nets. In *ICDAR*. IEEE, 2011.

[Neyshabur and Panigrahy, 2013] Behnam Neyshabur and Rina Panigrahy. Sparse matrix factorization. *arXiv preprint arXiv:1311.3315*, 2013.

[Olshausen and Field, 1997] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[Ren and Wu, 2014] Yuanfang Ren and Yan Wu. Convolutional deep belief networks for feature extraction of eeg signal. In *IJCNN*. IEEE, 2014.

[Rubinstein *et al.*, 2010] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Double sparsity: Learning sparse dictionaries for sparse signal approximation. *IEEE Transactions on signal processing*, 58(3):1553–1564, 2010.

[Schlögl, 2003] Alois Schlögl. Outcome of the bci-competition 2003 on the graz data set. *Berlin, Germany: Graz University of Technology*, 2003.

[Sprechmann *et al.*, 2015] Pablo Sprechmann, Alexander Bronstein, and Guillermo Sapiro. Learning efficient sparse and low rank models. *IEEE TPAMI*, 2015.

[Sutskever *et al.*, 2013] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[Tabar and Halici, 2016] Yousef Rezaei Tabar and Ugur Halici. A novel deep learning approach for classification of eeg motor imagery signals. *Journal of Neural Engineering*, 14(1):016003, 2016.

[Wan *et al.*, 2013] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[Wang *et al.*, 2016a] Zhangyang Wang, Shiyu Chang, Jiayu Zhou, Meng Wang, and Thomas S Huang. Learning a task-specific deep architecture for clustering. In *SDM*, 2016.

[Wang *et al.*, 2016b] Zhangyang Wang, Qing Ling, and Thomas Huang. Learning deep ℓ_0 encoders. *AAAI*, 2016.

[Wang *et al.*, 2016c] Zhangyang Wang, Ding Liu, Shiyu Chang, Qing Ling, Yingzhen Yang, and Thomas S Huang. D3: Deep dual-domain based fast restoration of jpeg-compressed images. In *IEEE CVPR*, 2016.

[Wang *et al.*, 2016d] Zhangyang Wang, Yingzhen Yang, Shiyu Chang, Qing Ling, and Thomas S Huang. Learning a deep l? encoder for hashing. In *IJCAI*, 2016.

[Wolpaw *et al.*, 2000] Jonathan R Wolpaw, Niels Birbaumer, William J Heetderks, Dennis J McFarland, P Hunter Peckham, Gerwin Schalk, Emanuel Donchin, Louis A Quatrano, Charles J Robinson, Theresa M Vaughan, et al. Brain-computer interface technology: a review of the first international meeting. *IEEE transactions on rehabilitation engineering*, 8(2):164–173, 2000.

[Xin *et al.*, 2016] Bo Xin, Yizhou Wang, Wen Gao, and David Wipf. Maximal sparsity with deep networks? *NIPS*, 2016.