Learning Deep ℓ_0 Encoders

Zhangyang Wang †, Qing Ling ‡, Thomas S. Huang †
†Beckman Institute, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
‡Department of Automation, University of Science and Technology of China, Hefei, 230027, China

Abstract

Despite its nonconvex nature, ℓ_0 sparse approximation is desirable in many theoretical and application cases. We study the ℓ_0 sparse approximation problem with the tool of deep learning, by proposing Deep ℓ_0 Encoders. Two typical forms, the ℓ_0 regularized problem and the M-sparse problem, are investigated. Based on solid iterative algorithms, we model them as feed-forward neural networks, through introducing novel neurons and pooling functions. Enforcing such structural priors acts as an effective network regularization. The deep encoders also enjoy faster inference, larger learning capacity, and better scalability compared to conventional sparse coding solutions. Furthermore, under task-driven losses, the models can be conveniently optimized from end to end. Numerical results demonstrate the impressive performances of the proposed encoders.

Dedication

Zhangyang and Qing would like to dedicate the paper to their friend, **Mr. Yuan Song** (10/09/1984 - 07/13/2015).

Introduction

Sparse signal approximation has gained popularity over the last decade. The sparse approximation model suggests that a natural signal could be compactly approximated, by only a few atoms out of a properly given dictionary, where the weights associated with the dictionary atoms are called the sparse codes. Proven to be both robust to noise and scalable to high dimensional data, sparse codes are known as powerful features, and benefit a wide range of signal processing applications, such as source coding (Donoho et al. 1998), denoising (Donoho 1995), source separation (Davies and Mitianoudis 2004), pattern classification (Wright et al. 2009), and clustering (Cheng et al. 2010).

We are particularly interested in the ℓ_0-based sparse approximation problem, which is the fundamental formulation of sparse coding (Donoho and Elad 2003). The nonconvex ℓ_0 problem is intractable and often instead attacked by minimizing surrogate measures, such as the ℓ_1-norm, which leads to more tractable computational methods. However, it has been both theoretically and practically discovered that solving ℓ_0 sparse approximation is still preferable in many cases.

More recently, deep learning has attracted great attentions in many feature learning problems (Krizhevsky, Sutskever, and Hinton 2012). The advantages of deep learning lie in its composition of multiple non-linear transformations to yield more abstract and descriptive embedding representations. With the aid of gradient descent, it also scales linearly in time and space with the number of train samples.

It has been noticed that sparse approximation and deep learning bear certain connections (Gregor and LeCun 2010). Their similar methodology has been lately exploited in (Hershey, Roux, and Weninger 2014), (Sprechmann et al. 2013), (Sprechmann, Bronstein, and Sapiro 2015). By turning sparse coding models into deep networks, one may expect faster inference, larger learning capacity, and better scalability. The network formulation also facilitates the integration of task-driven optimization.

In this paper, we investigate two typical forms of ℓ_0-based sparse approximation problems: the ℓ_0 regularized problem, and the M-sparse problem. Based on solid iterative algorithms (Blumensath and Davies 2008), we formulate them as feed-forward neural networks (Gregor and LeCun 2010), called **Deep ℓ_0 Encoders**, through introducing novel neurons and pooling functions. We study their applications in image classification and clustering; in both cases the models are optimized in a task-driven, end-to-end manner. Impressive performances are observed in numerical experiments.

Related Work

ℓ_0 and ℓ_1-based Sparse Approximations

Finding the sparsest, or minimum ℓ_0-norm, representation of a signal given a dictionary of basis atoms is an important problem in many application domains. Consider a data sample $\mathbf{x} \in R^{m\times 1}$, that is encoded into its sparse code $\mathbf{a} \in R^{p\times 1}$ using a learned dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p]$, where $\mathbf{d}_i \in R^{m\times 1}, i = 1, 2, \cdots, p$ are the learned atoms. The sparse codes are obtained by solving the ℓ_0 **regularized problem** (λ is a constant):

$$\mathbf{a} = \arg\min_{\mathbf{a}} \tfrac{1}{2}||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 + \lambda||\mathbf{a}||_0. \qquad (1)$$

Alternatively, one could explicitly impose constraints on the number of non-zero coefficients of the solution, by solving the M**-sparse problem**:

$$\mathbf{a} = \arg\min_{\mathbf{a}} ||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 \qquad s.t. \quad ||\mathbf{a}||_0 \leq M \qquad (2)$$

Unfortunately, these optimization problems are often intractable because there is a combinatorial increase in the number of local minima as the number of the candidate basis vectors increases. One potential remedy is to employ a convex surrogate measure, such as the ℓ_1-norm, in place of the ℓ_0-norm that leads to a more tractable optimization problem. For example, (1) could be relaxed as:

$$\mathbf{a} = \arg\min_{\mathbf{a}} \tfrac{1}{2}||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 + \lambda||\mathbf{a}||_1. \tag{3}$$

It creates a unimodal optimization problem that can be solved via linear programming techniques. The downside is that we have now introduced a mismatch between the ultimate goal and the objective function (Wipf and Rao 2004). Under certain conditions, the minimum ℓ_1-norm solution equals to the minimum ℓ_0-norm one (Donoho and Elad 2003). But in practice, the ℓ_1 approximation is often used way beyond these conditions, and is thus quite heuristic. As a result, we often get a solution which is not exactly minimizing the original ℓ_0-norm.

That said, ℓ_1 approximation is found to work practically well for many sparse coding problems. Yet in certain applications, we intend to control the exact number of nonzero elements, such as basis selection (Wipf and Rao 2004), where ℓ_0 approximation is indispensable. Beyond that, ℓ_0-approximation are desirable for performance concerns in many ways. In compressive sensing literature, empirical evidence (Candes, Wakin, and Boyd 2008) suggested that using an iterative reweighted ℓ_1 scheme to approximate the ℓ_0 solution often improved the quality of signal recovery. In image enhancement, it was shown in (Yuan and Ghanem 2015) that ℓ_0 data fidelity was more suitable for reconstructing images corrupted with impulse noise. For the purpose of image smoothening, the authors of (Xu et al. 2011) utilized ℓ_0 gradient minimization to globally control how many non-zero gradients to approximate prominent structures in a structure-sparsity-management manner. Recent work (Wang, Wang, and Singh 2015) revealed that ℓ_0 sparse subspace clustering can completely characterize the set of minimal union-of-subspace structure, without additional separation conditions required by its ℓ_1 counterpart.

Network Implementation of ℓ_1-Approximation

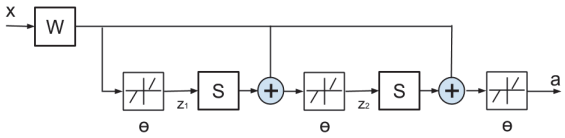

Figure 1: A LISTA network (Gregor and LeCun 2010) with two time-unfolded stages.

In (Gregor and LeCun 2010), a feed-forward neural network, as illustrated in Fig. 1, was proposed to efficiently approximate the ℓ_1-based sparse code $\mathbf{a}$ of the input signal $\mathbf{x}$; the sparse code a is obtained by solving (3) for a given dictionary $\mathbf{D}$ in advance. The network has a finite number of stages, each of which updates the intermediate sparse code $\mathbf{z}^k$ (k = 1, 2) according to

$$\mathbf{z}^{k+1} = s_\theta(\mathbf{W}\mathbf{x} + \mathbf{S}\mathbf{z}^k), \tag{4}$$

where s_θ is an element-wise shrinkage function ($\mathbf{u}$ is a vector and $\mathbf{u}_i$ is its i-th element, $i = 1, 2, ..., p$):

$$[s_\theta(\mathbf{u})]_i = \text{sign}(\mathbf{u}_i)(|\mathbf{u}_i| - \theta_i)_+. \tag{5}$$

The parameterized encoder, named learned ISTA (LISTA), is a natural network implementation of the iterative shrinkage and thresholding algorithm (ISTA). LISTA learned all its parameters $\mathbf{W}$, $\mathbf{S}$ and θ from training data using a back-propagation algorithm (LeCun et al. 2012). In this way, a good approximation of the underlying sparse code can be obtained after a fixed small number of stages.

In (Sprechmann et al. 2013), the authors leveraged a similar idea on fast trainable regressors and constructed feed-forward network approximations of the learned sparse models. Such a process-centric view was later extended in (Sprechmann, Bronstein, and Sapiro 2015) to develop a principled process of learned deterministic fixed-complexity pursuits, in lieu of iterative proximal gradient descent algorithms, for structured sparse and robust low rank models. Very recently, (Hershey, Roux, and Weninger 2014) further summarized the methodology of the problem-level and model-based "deep unfolding", and developed new architectures as inference algorithms for both Markov random fields and non-negative matrix factorization. Our work shares the similar spirit with those prior wisdoms, yet studies the unexplored ℓ_0 problems with further insights obtained.

Deep ℓ_0 Encoders

Deep ℓ_0-Regularized Encoder

To solve the optimization problem in (1), an iterative hard-thresholding (IHT) algorithm was derived in (Blumensath and Davies 2008):

$$\mathbf{a}^{k+1} = h_{\lambda^{0.5}}(\mathbf{a}^k + \mathbf{D}^\mathbf{T}(\mathbf{x} - \mathbf{D}\mathbf{a}^k)), \tag{6}$$

where $\mathbf{a}^k$ denotes the intermediate result of the k-th iteration, and h_θ is an element-wise **hard thresholding** operator:

$$[h_{\lambda^{0.5}}(\mathbf{u})]_i = \begin{cases} 0 & \text{if} \quad |\mathbf{u}_i| < \lambda^{0.5} \\ \mathbf{u}_i & \text{if} \quad |\mathbf{u}_i| \geq \lambda^{0.5} \end{cases} \tag{7}$$

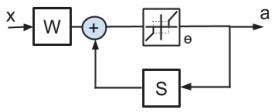

Figure 2: The block diagram of solving (6).

.

Eqn. (6) could be alternatively rewritten as:

$$\begin{aligned} &\mathbf{a}^{k+1} = h_\theta(\mathbf{W}\mathbf{x} + \mathbf{S}\mathbf{a}^k), \\ &\mathbf{W} = \mathbf{D}^T, \mathbf{S} = \mathbf{I} - \mathbf{D}^T\mathbf{D}, \theta = \lambda^{0.5}, \end{aligned} \tag{8}$$

and expressed as the block diagram in Fig. 2, which outlines a recurrent network form of solving (6).

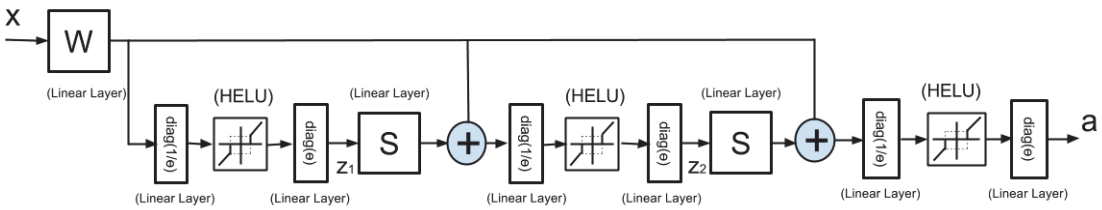

Figure 3: Deep ℓ_0-Regularized Encoder, with two time-unfolded stages.

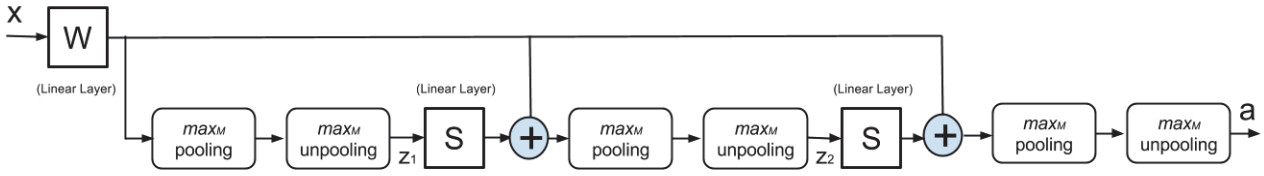

Figure 4: Deep M-sparse Encoder, with two time-unfolded stages.

By time-unfolding and truncating Fig. 2 to a fixed number of K iterations ($K = 2$ in this paper by default)[1], we obtain a feed-forward network structure in Fig. 3, where $\mathbf{W}$, $\mathbf{S}$ and θ are shared among both stages, named **Deep ℓ_0-Regularized Encoder**, Furthermore, $\mathbf{W}$, $\mathbf{S}$ and θ are all to be learnt, instead of being directly constructed from any pre-computed $\mathbf{D}$. Although the equations in (8) do not directly apply any more to solving the Deep ℓ_0-Regularized Encoder, they can usually serve as a high-quality initialization of the latter.

Note that the activation thresholds θ are less straightforward to update. We rewrite (5) as: $[h_\theta(\mathbf{u})]_i = \theta_i h_1(\mathbf{u}_i/\theta_i)$. It indicates that the original neuron with trainable thresholds can be decomposed into two linear scaling layers, plus a unit-hard-threshold neuron, the latter of which is called Hard thrEsholding Linear Unit (**HELU**) by us. The weights of the two scaling layers are diagonal matrices defined by θ and its element-wise reciprocal, respectively.

Discussion on HELU While being inspired by LISTA, the differentiating point of Deep ℓ_0-Regularized Encoder lies in the HELU neuron. Compared to classical neuron functions such as logistic, sigmoid, and ReLU (Mhaskar and Micchelli 1994), as well as the soft shrinkage and thresholding operation (5) in LISTA, HELU does not penalize large values, yet enforces strong (in theory infinite) penalty over small values. As such, HELU tends to produce highly sparse solutions.

The neuron form of LISTA (5) could be viewed as a double-sided and translated variant of ReLU, which is continuous and piecewise linear. In contrast, HELU is a **discontinuous** function that rarely occurs in existing deep network neurons. As pointed out by (Hornik, Stinchcombe, and White 1989), HELU has countably many discontinuities and is thus (Borel) measurable, in which case the universal approximation capability of the network is not compromised. However, experiments remind us that the algorithmic learnability with such discontinuous neurons (using popular first-order methods) is in question, and the training is in general hard. For computation concerns, we replace HELU with the following continuous and piecewise linear function HELU_σ, during network training:

$$[\text{HELU}_\sigma(\mathbf{u})]_i = \begin{cases} 0 & \text{if} \quad |\mathbf{u}_i| \leq 1-\sigma \\ \frac{(\mathbf{u}_i - 1 + \sigma)}{\sigma} & \text{if} \quad 1-\sigma < \mathbf{u}_i < 1 \\ \frac{(\mathbf{u}_i + 1 - \sigma)}{\sigma} & \text{if} \quad -1 < \mathbf{u}_i < \sigma - 1 \\ \mathbf{u}_i & \text{if} \quad |\mathbf{u}_i| \geq 1 \end{cases} \tag{9}$$

Obviously, HELU_σ becomes HELU when $\sigma \to 0$. To approximate HELU, we tend to choose very small σ, while avoiding putting the training ill-posed. As a practical strategy, we start with a moderate σ (0.2 by default), and divide it by 10 after each epoch. After several epoches, HELU_σ turns very close to the ideal HELU.

In (Rozell et al. 2008), the authors introduced an ideal hard thresholding function for solving sparse coding, whose formulation was close to HELU. Note that (Rozell et al. 2008) approximates the ideal function with a sigmoid function, which has connections with our HELU_σ approximation. In (Konda, Memisevic, and Krueger 2014), a similar truncated linear ReLU was utilized in the networks.

Deep M-Sparse ℓ_0 Encoder

Both the ℓ_0 regularized problem in (1) and Deep ℓ_0-Regularized Encoder have no explicit control on the sparsity level of the solution. One would therefore turn to the M-sparse problem in (2), and derive the following iterative algorithm (Blumensath and Davies 2008):

$$\mathbf{a}^{k+1} = h_M(\mathbf{a}^k + \mathbf{D}^\mathbf{T}(\mathbf{x} - \mathbf{D}\mathbf{a}^k)). \tag{10}$$

Eqn. (10) resembles (6), except that h_M is now a nonlinear operator retaining the M coefficients with the **top M-largest absolute values**. Following the same methodology

[1] We test larger K values (3 or 4). In several cases they do bring performance improvements, but add complexity too..

as in the previous section, the iterative form could be time-unfolded and truncated to the **Deep M-sparse Encoder**, as in Fig. 4. To deal with the h_M operation, we refer to the popular concepts of pooling and unpooling (Zeiler, Taylor, and Fergus 2011) in deep networks, and introduce the pairs of $\max_M$ pooling and unpooling, in Fig. 4.

Discussion on $\max_M$ pooling/unpooling Pooling is popular in convolutional networks to obtain translation-invariant features (Krizhevsky, Sutskever, and Hinton 2012). It is yet less common in other forms of deep networks (Gulcehre et al. 2014). The unpooling operation was introduced in (Zeiler, Taylor, and Fergus 2011) to insert the pooled values back to the appropriate locations of feature maps for reconstruction purposes.

In our proposed Deep M-sparse Encoder, the pooling and unpooling operation pair is used to construct a projection from R^m to its subset $S : \{s \in R^m | ||s||_0 \leq M\}$. The $\max_M$ pooling and unpooling functions are intuitively defined as:

$$\begin{array}{l}[\mathbf{p}_M, \mathbf{idx}_M] = \max_M.\text{pooling}(\mathbf{u}) \\ \mathbf{u}_M = \max_M.\text{unpooling}(\mathbf{p}_M, \mathbf{idx}_M)\end{array} \tag{11}$$

For each input $\mathbf{u}$, the *pooled map* $\mathbf{p}_M$ records the top M-largest values (irrespective of sign), and the *switch* $\mathbf{idx}_M$ records their locations. The corresponding unpooling operation takes the elements in $\mathbf{p}_M$ and places them in $\mathbf{u}_M$ at the locations specified by $\mathbf{idx}_M$, the remaining elements being set to zero. The resulting $\mathbf{u}_M$ is of the same dimension as $\mathbf{u}$ but has exactly no more than M non-zero elements. In back propagation, each position in $\mathbf{idx}_M$ is propagated with the entire error signal.

Theoretical Properties

It is showed in (Blumensath and Davies 2008) that the iterative algorithms in both (6) and (10) are guaranteed not to increase the cost functions. Under mild conditions, their targeted fixed points are local minima of the original problems. As the next step after the time truncation, the deep encoder models are to be solved by the stochastic gradient descent (SGD) algorithm, which converges to stationary points under a few stricter assumptions than ones satisfied in this paper (Bottou 2010) [2]. However, the entanglement of the iterative algorithms and the SGD algorithm makes the overall convergence analysis a serious hardship.

One must emphasize that in each step, the back propagation procedure requires only operations of order O(p) (Gregor and LeCun 2010). The training algorithm takes O(Cnp) time (C is the constant absorbing epochs, stage numbers, etc). The testing process is purely feed-forward and is therefore dramatically faster than traditional inference methods by solving (1) or (2). SGD is also easy to be parallelized.

Task-Driven Optimization

It is often desirable to jointly optimize the learned sparse code features and the targeted task so that they mutually reinforce each other. The authors of (Jiang, Lin, and Davis 2011) associated label information with each dictionary item by adding discriminable regularization terms to the objective. Recent work (Mairal, Bach, and Ponce 2012), (Wang et al. 2015a) developed task-driven sparse coding via bi-level optimization models, where (ℓ_1-based) sparse coding is formulated as the lower-level constraint while a task-oriented cost function is minimized as its upper-level objective. The above approaches in sparse coding are complicated and computationally expensive. It is much more convenient to implement end-to-end task-driven training in deep architectures, by concatenating the proposed deep encoders with certain task-driven loss functions.

In this paper, we mainly discuss two tasks: classification and clustering, while being aware of other immediate extensions, such as semi-supervised learning. Assuming K classes (or clusters), and $\boldsymbol{\omega} = [\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}^k]$ as the set of parameters of the loss function, where $\boldsymbol{\omega}_i$ corresponds to the j-th class (cluster), $j = 1, 2, ..., K$. For the **classification** case, one natural choice is the well-known softmax loss function. For the **clustering** case, since the true cluster label of each $\mathbf{x}$ is unknown, we define the predicted confidence probability p_j that sample $\mathbf{x}$ belongs to cluster j, as the likelihood of softmax regression:

$$p_j = p(j|\boldsymbol{\omega}, \boldsymbol{a}) = \frac{e^{-\boldsymbol{\omega}_j^T \mathbf{a}}}{\sum_{l=1}^{K} e^{-\boldsymbol{\omega}_l^T \mathbf{a}}}. \tag{12}$$

The predicted cluster label of $\mathbf{a}$ is the cluster j where it achieves the largest p_j.

Experiment

Implementation

Two proposed deep ℓ_0 encoders are implemented with the CUDA ConvNet package (Krizhevsky, Sutskever, and Hinton 2012). We use a constant learning rate of 0.01 with no momentum, and a batch size of 128. In practice, given that the model is well initialized, the training takes approximately 1 hour on the MNIST dataset, on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. It is also observed that the training efficiency of our model scales approximately linearly with the size of data.

While many neural networks train well with random initializations without pre-training, given that the training data is sufficient, it has been discovered that poorly initialized networks can hamper the effectiveness of first-order methods (e.g., SGD) (Sutskever et al. 2013). For the proposed models, it is however much easier to initialize the model in the right regime, benefiting from the analytical relationships between sparse coding and network hyperparameters in (8).

Simulation on ℓ_0 Sparse Approximation

We first compare the performance of different methods on ℓ_0 sparse code approximation. The first 60,000 samples of the MNIST dataset are used for training and the last 10,000 for testing. Each patch is resized to 16×16 and then pre-processed to remove its mean and normalize its variance. The patches with small standard deviations are discarded. A sparsity coefficient λ = 0.5 is used in (1), and the sparsity level M = 32 is fixed in (2). The sparse code dimension (dictionary size) p is to be varied.

[2] As a typical case, we use SGD in a setting where it is not guaranteed to converge in theory, but behaves well in practice.

Table 1: Prediction error (%) comparison of all methods on solving the ℓ_0-regularized problem (1)

p	128	256	512
Iterative (2 iterations)	17.52	18.73	22.40
Iterative (5 iterations)	8.14	6.75	9.37
Iterative (10 iterations)	3.55	4.33	4.08
Baseline Encoder	8.94	8.76	10.17
Deep ℓ_0-Regularized Encoder	0.92	0.91	0.81

Our prediction task resembles the setup in (Gregor and LeCun 2010): first learning a dictionary from training data, following by solving sparse approximation (3) with respect to the dictionary, and finally training the network as a regressor from input samples to the solved sparse codes. The only major difference here lies in that unlike the ℓ_1-based problems, the non-convex ℓ_0-based minimization could only reach a (non-unique) local minimum. To improve stability, we first solve the ℓ_1 problems to obtain a good initialization for ℓ_0 problems, and then run the iterative algorithms (6) or (10) until convergence. The obtained sparse codes are called "optimal codes" hereinafter and used in both training and testing evaluation (as "groundtruth"). One must keep in mind that we are not seeking to produce approximate sparse code for all possible input vectors, but only for *input vectors drawn from the same distribution as our training samples*.

Table 2: Prediction error (%) comparison of all methods on solving the M-sparse problem (2)

p	128	256	512
Iterative (2 iterations)	17.23	19.27	19.31
Iterative (5 iterations)	10.84	12.52	12.40
Iterative (10 iterations)	5.67	5.44	5.20
Baseline Encoder	14.04	16.76	12.86
Deep M-Sparse Encoder	2.94	2.87	3.29

Table 3: Averaged non-zero support error comparison of all methods on solving the M-sparse problem (2)

p	128	256	512
Iterative (2 iterations)	10.8	13.4	13.2
Iterative (5 iterations)	6.1	8.0	8.8
Iterative (10 iterations)	4.6	5.6	5.3
Deep M-Sparse Encoder	2.2	2.7	2.7

We compare the proposed deep ℓ_0 encoders with the iterative algorithms under different number of iterations. In addition, we include a *baseline encoder* into comparison, which is a fully-connected feed-forward network, consisting of three hidden layers of dimension p with ReLu neurons. The baseline encoder thus has the same parameter capacity as deep ℓ_0 encoders[3]. We apply dropout to the baseline encoders, with the probabilities of retaining the units being 0.9, 0.9, and 0.5. The proposed encoders do not apply dropout.

The deep ℓ_0 encoders and the baseline encoder are first trained, and all are then evaluated on the testing set. We calculate the total prediction errors, i.e., the normalized squared errors between the optimal codes and the predicted codes, as in Tables 1 and 2. For the M-sparse case, we also compare their recovery of non-zero supports in Table 3, by counting the mismatched nonzero element locations between optimal and predicted codes (averaged on all samples). Immediate conclusions from the numerical results are as follows:

- The proposed deep encoders have outstanding generalization performances, thanks to the effective regularization brought by their architectures, which are derived from specific problem formulations (i.e., (1) and (2)) as priors. The "general-architecture" baseline encoders, which have the same parameter complexity, appear to overfit the training set and generalize much worse.
- While the deep encoders only unfold two stages, they outperforms their iterative counterparts even when the later ones have passed 10 iterations. Meanwhile, the former enjoy much faster inference as being feed-forward.
- The Deep ℓ_0-Regularized Encoder obtains a particularly low prediction error. It is interpretable that while the iterative algorithm has to work with a fixed λ, the Deep ℓ_0-Regularized Encoder is capable of "fine-tuning" this hyper-parameter automatically (after diag(θ) is initialized from λ), by exploring the training data structure.
- The Deep M-Sparse Encoder is able to find the nonzero support with high accuracy.

Applications on Classification

Since the task-driven models are trained from end to end, **no pre-computation of a is needed**. For classification, we evaluate our methods on the MNIST dataset, and the AVIRIS Indiana Pines hyperspectral image dataset (see (Wang, Nasrabadi, and Huang 2015) for details). We compare our two proposed deep encoders with two competitive sparse coding-based methods: 1) task-driven sparse coding (TDSC) in (Mairal, Bach, and Ponce 2012), with the original setting followed and all parameters carefully tuned; 2) a pre-trained LISTA followed by supervised tuning with softmax loss. Note that for Deep M-Sparse Encoder, M is not known in advance and has to be tuned. To our surprise, the fine-tuning of M is likely to improve the performances significantly, which is analyzed next. The overall error rates are compared in Tables 4 and 5.

In general, the proposed deep ℓ_0 encoders provide superior results to the deep ℓ_1-based method (tuned LISTA). TDSC also generates competitive results, but at the cost of the high complexity for inference, i.e., solving conventional sparse coding. It is of particular interest to us that when supplied with specific M values, the Deep M-Sparse encoder can generate remarkably improved results [4]. Especially in Table 5, when M = 10, the error rate is around 1.5% lower

[3] except for the "diag(θ)" layers in Fig. 3, each of which contains only p free parameters.

[4] To get a good estimate of M, one might first try to perform (unsupervised) sparse coding on a subset of samples.

Table 4: Classification error rate (%) comparison of all methods on the MNIST dataset

p	128	256	512
TDSC	0.71	0.55	0.53
Tuned LISTA	0.74	0.62	0.57
Deep ℓ_0-Regularized	0.72	0.58	0.52
Deep M-Sparse (M = 10)	0.72	0.57	0.53
Deep M-Sparse (M = 20)	0.69	0.54	0.51
Deep M-Sparse (M = 30)	0.73	0.57	0.52

Table 5: Classification error rate (%) comparison of all methods on the AVIRIS Indiana Pines dataset

p	128	256	512
TDSC	15.55	15.27	15.21
Tuned LISTA	16.12	16.05	15.97
Deep ℓ_0-Regularized	15.20	15.07	15.01
Deep M-Sparse (M = 10)	13.77	13.56	13.52
Deep M-Sparse (M = 20)	14.67	14.23	14.07
Deep M-Sparse (M = 30)	15.14	15.02	15.00

than that of M = 30. Note that in the AVIRIS Indiana Pines dataset, the training data volume is much smaller than that of MNIST. In that way, we conjecture that it might not be sufficiently effective to depend the training process fully on data; instead, to craft a stronger sparsity prior by smaller M could help learn more discriminative features[5]. Such a behavior provides us with a important hint to **impose suitable structural priors to deep networks.**

Applications on Clustering

For clustering, we evaluate our methods on the COIL 20 and the CMU PIE dataset (Sim, Baker, and Bsat 2002). Two state-of-the-art methods to compare are the jointly optimized sparse coding and clustering method proposed in (Wang et al. 2015a), as well as the graph-regularized deep clustering method in (Wang et al. 2015b)[6]. The overall error rates are compared in Tables 6 and 7.

Note that the method in (Wang et al. 2015b) incorporated Laplacian regularization as an additional prior while the others not. It is thus no wonder that this method often performs better than others. Even without any graph information utilized, the proposed deep encoders are able to obtain very close performances, and outperforms (Wang et al. 2015b) in certain cases. On the COIL 20 dataset, the lowest error rate is reached by the Deep M-Sparse (M = 10) Encoder, when p = 512, followed by the Deep ℓ_0-Regularized Encoder.

[5]Interestingly, there are a total of 16 classes in the AVIRIS Indiana Pines dataset When p = 128, each class has on average 8 "atoms" for class-specific representation. Therefore M = 10 approximately coincides with the sparse representation classification (SRC) principle (Wang, Nasrabadi, and Huang 2015) of forcing sparse codes to be compactly focused on one class of atoms.

[6]Both papers train their model under both soft-max and max-margin type losses. To ensure fair comparison, we adopt the former, with the same form of loss function as ours.

Table 6: Clustering error rate (%) comparison of all methods on the COIL 20 dataset

p	128	256	512
(Wang et al. 2015a)	17.75	17.14	17.15
(Wang et al. 2015b)	14.47	14.17	14.08
Deep ℓ_0-Regularized	14.52	14.27	14.06
Deep M-Sparse (M = 10)	14.59	14.25	14.03
Deep M-Sparse (M = 20)	14.84	14.33	14.15
Deep M-Sparse (M = 30)	14.77	14.37	14.12

Table 7: Clustering error rate (%) comparison of all methods on the CMU PIE dataset

p	128	256	512
(Wang et al. 2015a)	17.50	17.26	17.20
(Wang et al. 2015b)	16.14	15.58	15.09
Deep ℓ_0-Regularized	16.08	15.72	15.41
Deep M-Sparse (M = 10)	16.77	16.46	16.02
Deep M-Sparse (M = 20)	16.44	16.23	16.05
Deep M-Sparse (M = 30)	16.46	16.17	16.01

On the CMU PIE dataset, the Deep ℓ_0-Regularized Encoder leads to competitive accuracies with (Wang et al. 2015b), and outperforms all Deep M-Sparse Encoders with noticeable margins, which is different from other cases. Pervious work discovered that sparse approximations over CMU PIE had significant errors (Yang, Yu, and Huang 2010), which is also verified by us. Therefore, hardcoding exact sparsity could even hamper the model performance.

Remark: From those experiments, we gain additional insights in designing deep architectures:

- If one expects the model to explore the data structure by itself, and provided that there is sufficient training data, then the Deep ℓ_0-Regularized Encoder (and its peers) might be preferred as its all parameters, including the desired sparsity, are fully learnable from the data.
- If one has certain correct prior knowledge of the data structure, including but not limited to the exact sparsity level, one should choose Deep M-Sparse Encoder, or other models of its type that are designed to maximally enforce that prior. The methodology could be especially useful when the training data is less than sufficient.

We hope the above insights could be of reference to many other deep learning models.

Conclusion

We propose Deep ℓ_0 Encoders to solve the ℓ_0 sparse approximation problem. Rooted in solid iterative algorithms, the deep ℓ_0 regularized encoder and deep M-sparse encoder are developed, each designed to solve one typical formulation, accompanied with the introduction of the novel HELU neuron and $\max_M$ pooling/unpooling. When applied to specific tasks of classification and clustering, the models are optimized in an end-to-end manner. The latest deep learning tools enable us to solve them in a highly effective and

efficient fashion. They not only provide us with impressive performances in numerical experiments, but also inspire us with important insights into designing deep models.

References

Blumensath, T., and Davies, M. E. 2008. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications* 14(5-6):629–654.

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer. 177–186.

Candes, E. J.; Wakin, M. B.; and Boyd, S. P. 2008. Enhancing sparsity by reweighted l1 minimization. *Journal of Fourier analysis and applications* 14(5-6):877–905.

Cheng, B.; Yang, J.; Yan, S.; Fu, Y.; and Huang, T. S. 2010. Learning with l1 graph for image analysis. *TIP* 19(4).

Davies, M., and Mitianoudis, N. 2004. Simple mixture model for sparse overcomplete ica. *IEE Proceedings-Vision, Image and Signal Processing* 151(1):35–43.

Donoho, D. L., and Elad, M. 2003. Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization. *Proceedings of the National Academy of Sciences* 100(5):2197–2202.

Donoho, D. L.; Vetterli, M.; DeVore, R. A.; and Daubechies, I. 1998. Data compression and harmonic analysis. *Information Theory, IEEE Transactions on* 44(6):2435–2476.

Donoho, D. L. 1995. De-noising by soft-thresholding. *Information Theory, IEEE Transactions on* 41(3):613–627.

Gregor, K., and LeCun, Y. 2010. Learning fast approximations of sparse coding. In *ICML*, 399–406.

Gulcehre, C.; Cho, K.; Pascanu, R.; and Bengio, Y. 2014. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 530–546.

Hershey, J. R.; Roux, J. L.; and Weninger, F. 2014. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2(5):359–366.

Jiang, Z.; Lin, Z.; and Davis, L. S. 2011. Learning a discriminative dictionary for sparse coding via label consistent k-svd. In *CVPR*, 1697–1704. IEEE.

Konda, K.; Memisevic, R.; and Krueger, D. 2014. Zero-bias autoencoders and the benefits of co-adapting features. *arXiv preprint arXiv:1402.3337*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

LeCun, Y. A.; Bottou, L.; Orr, G. B.; and Müller, K.-R. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer. 9–48.

Mairal, J.; Bach, F.; and Ponce, J. 2012. Task-driven dictionary learning. *TPAMI* 34(4):791–804.

Mhaskar, H. N., and Micchelli, C. A. 1994. How to choose an activation function. In *NIPS*, 319–326.

Rozell, C. J.; Johnson, D. H.; Baraniuk, R. G.; and Olshausen, B. A. 2008. Sparse coding via thresholding and local competition in neural circuits. *Neural computation* 20(10):2526–2563.

Sim, T.; Baker, S.; and Bsat, M. 2002. The cmu pose, illumination, and expression (pie) database. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, 46–51. IEEE.

Sprechmann, P.; Litman, R.; Yakar, T. B.; Bronstein, A. M.; and Sapiro, G. 2013. Supervised sparse analysis and synthesis operators. In *NIPS*, 908–916.

Sprechmann, P.; Bronstein, A.; and Sapiro, G. 2015. Learning efficient sparse and low rank models. *TPAMI*.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *ICML*, 1139–1147.

Wang, Z.; Yang, Y.; Chang, S.; Li, J.; Fong, S.; and Huang, T. S. 2015a. A joint optimization framework of sparse coding and discriminative clustering. In *IJCAI*.

Wang, Z.; Chang, S.; Zhou, J.; Wang, M.; and Huang, T. S. 2015b. Learning a task-specific deep architecture for clustering. In *arXiv preprint arXiv:1509.00151*.

Wang, Z.; Nasrabadi, N. M.; and Huang, T. S. 2015. Semisupervised hyperspectral classification using task-driven dictionary learning with laplacian regularization. *TGRS* 53(3):1161–1173.

Wang, Y.; Wang, Y.-X.; and Singh, A. 2015. Clustering consistent sparse subspace clustering. *arXiv preprint arXiv:1504.01046*.

Wipf, D. P., and Rao, B. D. 2004. l0-norm minimization for basis selection. In *NIPS*, 1513–1520.

Wright, J.; Yang, A. Y.; Ganesh, A.; Sastry, S. S.; and Ma, Y. 2009. Robust face recognition via sparse representation. *TPAMI* 31(2):210–227.

Xu, L.; Lu, C.; Xu, Y.; and Jia, J. 2011. Image smoothing via l 0 gradient minimization. In *TOG*, volume 30, 174. ACM.

Yang, J.; Yu, K.; and Huang, T. 2010. Supervised translation-invariant sparse coding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 3517–3524. IEEE.

Yuan, G., and Ghanem, B. 2015. L0tv: A new method for image restoration in the presence of impulse noise.

Zeiler, M. D.; Taylor, G. W.; and Fergus, R. 2011. Adaptive deconvolutional networks for mid and high level feature learning. In *ICCV*, 2018–2025. IEEE.