Published as: *Neural Comput*. 2012 November ; 24(11): 2852–2872.

A network of spiking neurons for computing sparse representations in an energy efficient way

Tao Hu[1], **Alexander Genkin**[2], and **Dmitri B. Chklovskii**[1]

Tao Hu: hut@janelia.hhmi.org; Alexander Genkin: alexgenkin@iname.com; Dmitri B. Chklovskii: mitya@janelia.hhmi.org

[1]Howard Hughes Medical Institute, Janelia Farm Research Campus, Ashburn, VA 20147, USA

[2]AVG Consulting, Brooklyn, NY, USA

Abstract

Computing sparse redundant representations is an important problem both in applied mathematics and neuroscience. In many applications, this problem must be solved in an energy efficient way. Here, we propose a hybrid distributed algorithm (HDA), which solves this problem on a network of simple nodes communicating via low-bandwidth channels. HDA nodes perform both gradient-descent-like steps on analog internal variables and coordinate-descent-like steps via quantized external variables communicated to each other. Interestingly, such operation is equivalent to a network of integrate-and-fire neurons, suggesting that HDA may serve as a model of neural computation. We compare the numerical performance of HDA with existing algorithms and show that in the asymptotic regime the representation error of HDA decays with time, t, as $1/t$. We show that HDA is stable against time-varying noise, specifically, the representation error decays as $1/\sqrt{t}$ for Gaussian white noise.

1 Introduction

Many natural signals can be represented as linear combinations of a few feature vectors (or elements) chosen from a redundant (or overcomplete) dictionary. Such representations are called sparse because most dictionary elements enter with zero coefficients. The importance of sparse representations has been long recognized in applied mathematics (Chen et al., 1998, Baraniuk, 2007) and in neuroscience, where electrophysiological recordings (DeWeese et al., 2003) and theoretical arguments (Attwell and Laughlin, 2001, Lennie, 2003) demonstrate that most neurons are silent at any given moment (Olshausen and Field, 1996, Gallant and Vinje, 2000, Olshausen and Field, 2004).

In applied mathematics, sparse representations lie at the heart of many important developments. In signal processing, such solutions serve as a foundation for basis pursuit (Chen et al., 1998) de-noising, compressive sensing (Baraniuk, 2007) and object recognition (Kavukcuoglu et al., 2010). In statistics, regularized multivariate regression algorithms, such as the Lasso (Tibshirani, 1996) or the elastic net (Zou and Hastie, 2005), rely on sparse representations to perform feature subset selection along with coefficient fitting. Given the importance of finding sparse representations, it is not surprising that many algorithms have been proposed for the task (Efron et al., 2004, Zou and Hastie, 2005, Friedman et al., 2007, Yin et al., 2008, Cai et al., 2009a, b, Li and Osher, 2009, Xiao, 2010). However, most algorithms are designed for CPU architectures and are computationally and energy intensive.

Given the ubiquity of sparse representations in neuroscience, how can neural networks generate sparse representations remains a central question. Building on the seminal work of Olshausen and Field (Olshausen and Field, 1996), Rozell et al. have proposed an algorithm

for sparse representations by neural networks called Local Competitive Algorithm (LCA) (Rozell et al., 2008). Such algorithm computes a sparse representation on a network of nodes that communicate analog variables with each other. Although a step towards biological realism, the LCA neglects the fact that most neurons communicate using action potentials (or spikes) - quantized all-or-none electrical signals. Although spiking neurons can communicate analog variables by firing rates, their punctuate nature leads to computational inferiority relative to pure analog unless the limit of large number of spikes is taken (Deneve and Boerlin, 2011, Shapero et al., 2011). However, this limit erases the advantage of spiking in terms of energy efficiency, an important consideration in brain design (Attwell and Laughlin, 2001, Laughlin and Sejnowski, 2003).

In this paper, we introduce an energy efficient algorithm called hybrid distributed algorithm (HDA), which computes sparse redundant representations on the architecture of (Rozell et al., 2008) but using neurons that spike. We demonstrate that such algorithm performs as well as the analog one, thus suggesting that spikes may not detrimentally affect computational capabilities of neural circuits. Moreover, HDA can serve as a plausible model of neural computation because local operations are described by the biologically inspired integrate-and-fire neurons (Koch, 1999, Dayan and Abbott, 2001). Other spiking neuron models have been proposed for sensory integration, working memory (Boerlin and Deneve, 2011) and implementing dynamical systems (Deneve and Boerlin, 2011, Shapero et al., 2011).

Because spiking communication requires smaller bandwidth, HDA may also be useful for sensor networks, which must discover sparse causes in distributed signals. In particular, large networks of small autonomous nodes are commonly deployed both for civilian and military applications, such as monitoring the motion of tornado or forest fires, tracking traffic conditions, security surveillance in shopping malls and parking facilities, locating and tracking enemy movements, detection of terrorist threats and attacks, (Tubaishat and Madria, 2003). The nodes of such networks use finite-life or slowly charging batteries and, hence, must operate under limited energy budget. Therefore, low-energy computations and limited bandwidth communication are two central design principles of such networks. Because correlations are often present among distributed sensor nodes, computing sparse redundant representations is an important task.

The paper is organized as follows. In §2 we describe the Bregman iteration method for computing sparse representations and briefly introduce two other distributed methods. We then consider a refined Bregman iteration method with coordinate descent modifications (§3) and continue in §4 by deriving our hybrid distributed algorithm. In §5 we prove the asymptotic performance guarantee of HDA, and demonstrate its numerical performance in §6. Finally, we conclude with the discussion of the advantages of HDA (§7).

2 Problem statement and existing distributed algorithms

A sparse solution $\boldsymbol{u} \in \mathbb{R}^n$ of the equation $\boldsymbol{Au} = \boldsymbol{f}$, where $\boldsymbol{f} \in \mathbb{R}^m$, and wide matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ ($n > m$) can be found by solving the following constrained optimization problem:

$$\min\|\boldsymbol{u}\|_1 \text{s.t.} \boldsymbol{Au}=\boldsymbol{f}, \quad (1)$$

which is known as basis pursuit (Chen et al., 1998). In practical applications, where $\boldsymbol{f}$ contains noise, one typically formulates the problem differently, in terms of an unconstrained optimization problem known as the Lasso (Tibshirani, 1996):

$$\min\frac{1}{2}\|\boldsymbol{Au}-\boldsymbol{f}\|_2^2+\lambda\|\boldsymbol{u}\|_1, \quad (2)$$

where λ is the regularization parameter which controls the trade-off between representation error and sparsity. The choice of regularization by l_1-norm assures that the problem both remains convex (Boyd and Vandenberghe, 2004, Dattorro, 2008, Bertsekas, 2009) and favors sparse solutions (Tibshirani, 1996, Chen et al., 1998). In this paper we introduce an energy efficient algorithm that searches for a solution to the constrained optimization problem (1) by taking steps towards solving a small number of unconstrained optimization problems (2). Our algorithm is closest to the family of algorithms called Bregman iterations (Yin et al., 2008, Cai et al., 2009a, b, Osher et al., 2010), which take their name from the replacement of the l_1-norm by its Bregman divergence (Bregman, 1967), $D(\boldsymbol{u}, \boldsymbol{u}^k) = \lambda|\boldsymbol{u}|_1 - \lambda|\boldsymbol{u}^k|_1 - \langle \boldsymbol{p}^k, \boldsymbol{u} - \boldsymbol{u}^k\rangle$, where $\boldsymbol{p}$ is a sub-gradient of $\lambda|\boldsymbol{u}|_1$ (Boyd and Vandenberghe, 2004).. The iterations start with $\boldsymbol{u}^0=\boldsymbol{p}^0=0$ and consist of two steps:

$$\boldsymbol{u}^{k+1}=\mathrm{argmin}_u E=\mathrm{argmin}_u\{\frac{1}{2}\|\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f}\|_2^2+\lambda\|\boldsymbol{u}\|_1-\lambda\|\boldsymbol{u}^k\|_1-\langle \boldsymbol{p}^k, \boldsymbol{u}-\boldsymbol{u}^k\rangle\}. \quad (3)$$

$$\boldsymbol{p}^{k+1}=\boldsymbol{p}^k-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{u}^{k+1}-\boldsymbol{f}). \quad (4)$$

Throughout the paper, we assume that $\boldsymbol{A}$ is column normalized, i.e. if $\boldsymbol{A}_i$ is the i-th column of $\boldsymbol{A}$, $\boldsymbol{A}_i^T\boldsymbol{A}_i=1$. Note that, because $n > m$, $\boldsymbol{A}$ defines a (redundant) frame. Moreover, we assume that $|\boldsymbol{A}_i^T\boldsymbol{A}_j|<1$ for any $i \neq j$.

A practical algorithm for solving (1) called linearized Bregman iterations (LBI) is derived by solving the optimization problem (3) approximately (Yin et al., 2008, Cai et al., 2009a, b). The square error term in Eq. (3) is replaced by its linear approximation $\langle \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f}), \boldsymbol{u}-\boldsymbol{u}^k\rangle$ around $\boldsymbol{u}^k$ and a proximity term $\frac{1}{2\delta}\|\boldsymbol{u}-\boldsymbol{u}^k\|_2^2$ is added to reflect the limited range of validity of the linear approximation. After some algebra the steps (3) and (4) reduce to the following two-step LBI (Yin et al., 2008, Cai et al., 2009a, b):

$$\nu^{k+1}=\nu^k-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{u}^k-\boldsymbol{f}), \quad (5)$$

$$\boldsymbol{u}^{k+1}=\delta\,\mathrm{shrink}(\nu^{k+1},\lambda), \quad (6)$$

where $\boldsymbol{\nu}^k = \boldsymbol{p}^k + \boldsymbol{u}^k/\delta$ and the component wise operation

$$\mathrm{shrink}(x,\lambda)=\begin{cases} x-\lambda, & \text{if } x>\lambda \\ 0, & \text{if } -\lambda<x<\lambda \\ x+\lambda, & \text{if } x<-\lambda \end{cases}$$ (Elad et al., 2007).

The LBI can be naturally implemented by a network of n parallel nodes, Figure 1, an architecture previously proposed to implement LCA (Rozell et al., 2008). Such a network combines feedforward projections, $\boldsymbol{A}^T$, and inhibitory lateral connections, $-\boldsymbol{A}^T\boldsymbol{A}$, which implement "explaining away" (Pearl, 1988). At every step, each node updates its component of the internal variable, $\boldsymbol{\nu}$, by adding the corresponding components of the feedforward signal, $\boldsymbol{A}^T\boldsymbol{f}$, and the broadcast external variable, $-\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{u}$. Then, each node computes the new value of its component in $\boldsymbol{u}$ by shrinking its component in $\boldsymbol{\nu}$. Another distributed algorithm called RDA (Xiao, 2010) can also be implemented by such a network.

Although LBI, LCA or RDA achieve sparse approximation of the incoming signal, implementing these algorithms in man-made or biological hardware using the network architecture of Fig. 1 would be challenging in practice. The reason is that all these

algorithms require real-time communication of analog variables, thus placing high demands on the energy consumption and bandwidth of lateral connections. Considering that the potential number of lateral connections is $O(n^2)$, and both volume and energy are often a limited resource in the brain (Attwell and Laughlin, 2001, Chklovskii et al., 2002, Laughlin and Sejnowski, 2003) and in sensor networks (Tubaishat and Madria, 2003) we search for a more efficient solution.

3 Bregman coordinate descent

In an attempt to find a distributed algorithm for solving (1) under bandwidth limitations, we explore a different strategy, called coordinate descent, where only one component of $\boldsymbol{u}$ is updated at a given iteration (Friedman et al., 2007). Inspired by (Li and Osher, 2009) we derive a novel Bregman coordinate descent algorithm. We start from (3) and rewrite the energy function on the right hand side by substituting matrix notation with explicit summation over vector components:

$$E=\frac{1}{2}\left\|\sum_{j=1}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f}\right\|_2^2+\lambda\sum_{j=1}^{n}\|u_j\|_1-\lambda\sum_{j=1}^{n}\left\|u_j^k\right\|_1-\sum_{j=1}^{n}p_j^k(u_j-u_j^k). \quad (7)$$

Assuming that in the (k+1)-th iteration, the i-th component of $\boldsymbol{u}$ is to be updated, and the values of all other components of $\boldsymbol{u}$ remain unchanged, then the updated value u_i' is obtained from

$$u_i'=\mathrm{argmin}_x E=\mathrm{argmin}_x\left\{\frac{1}{2}\left\|x\boldsymbol{A}_i+\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f}\right\|_2^2+\lambda\|u_i\|_1-p_iu_i\right\}, \quad (8)$$

where we denote the i-th component of $\boldsymbol{u}$ to be updated as x. In iteration (8) we drop terms independent of u_i and do not keep track of the iteration number k. The condition for the minimum in (8) is

$$\partial[\lambda\|u_i'\|_1]\ni-\boldsymbol{A}_i^T(u_i'\boldsymbol{A}_i+\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})+p_i, \quad (9)$$

where ∂ [.] designates a subdifferential (Boyd and Vandenberghe, 2004). Noticing $\boldsymbol{A}_i^T\boldsymbol{A}_i=1$ and $\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j=\boldsymbol{A}\boldsymbol{u}-u_i\boldsymbol{A}_i$, we rewrite (9) as

$$\partial[\lambda\|u_i'\|_1]\ni-u_i'+u_i-\boldsymbol{A}_i^T(\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f})+p_i, \quad (10)$$

From the optimality condition (10), we get the update formula of p_i (Yin et al., 2008),

$$p_i'+u_i'=p_i+u_i-\boldsymbol{A}_i^T(\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f}), \quad (11)$$

where $\partial[\lambda\|u_i'\|_1]\ni p_i'$. By defining $\nu_i=p_i+u_i$, we get:

$$\nu_i'=\nu_i-\boldsymbol{A}_i^T(\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f}). \quad (12)$$

Then we derive the update formula of u_i. Noticing

$$\begin{aligned}
&\|x\boldsymbol{A}_i+\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f}\|_2^2\\
&=x^2+2x\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})+\|\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f}\|_2^2\\
&=\|x+\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})\|_2^2\\
&-\|\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})\|_2^2\\
&+\|\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f}\|_2^2\\
&=\|x+\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})\|_2^2+const,
\end{aligned}\tag{13}$$

we rewrite (8) as

$$\begin{aligned}
u_i^{'}=&\operatorname{argmin}_x\left\{\frac{1}{2}\|x+\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})\|_2^2\right.\\
&+\lambda\|u_i\|_1-p_iu_i=\operatorname{argmin}_x\left\{\frac{1}{2}\|x-p_i+\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j-\boldsymbol{f})\|_2^2\right.\\
&\left.\left.+\lambda\|u_i\|_1\right\}\right.\\
&=\operatorname{shrink}(p_i\\
&-\boldsymbol{A}_i^T(\textstyle\sum_{j\neq i}^{n}u_j\boldsymbol{A}_j\\
&-\boldsymbol{f}),\lambda)=\operatorname{shrink}(p_i\\
&+u_i-\boldsymbol{A}_i^T(\boldsymbol{A}\boldsymbol{u}-\boldsymbol{f}),\lambda).
\end{aligned}\tag{14}$$

By substituting Eqns. (11) and (12) into (14), we get:

$$u_i^{'}=\operatorname{shrink}(\nu_i^{'},\lambda).\tag{15}$$

These iterations appear similar to that in LBI (5, 6), but are performed in a component-wise manner resulting in the following algorithm.

Algorithm 1

Bregman coordinate descent

Initialize: $\boldsymbol{\nu}=0$, $\boldsymbol{u}=0$

while "$\|\boldsymbol{f}-\boldsymbol{A}\boldsymbol{u}\|_2^2$ not converge" **do**

"choose $i\in\{1:n\}$"

$$\nu_i\leftarrow\nu_i-A_i^T(Au-f),\tag{16}$$

$$u_i\leftarrow\operatorname{shrink}(\nu_i,\lambda).\tag{17}$$

end while

In addition to specifying component-wise iterations in Algorithm 1, we must also specify the order in which the components of $\boldsymbol{u}$ are updated. Previous proposal include updating components sequentially based on the index i (Friedman et al., 2007, Genkin et al., 2007), randomly, or based on the gradient of the objective function (Li and Osher, 2009). In general, choosing i in a distributed architecture requires additional communication between nodes and, therefore, places additional demands on energy consumption and communication bandwidth.

4 Derivation of the Hybrid Distributed Algorithm (HDA)

Here, we present our central contribution, a distributed algorithm for solving (1), which has lower communication bandwidth requirements than the existing ones and does not require additional communication for determining the update order. We name our algorithm Hybrid Distributed Algorithm (HDA) because it combines a gradient-descent-like update of $\boldsymbol{\nu}$, as in Eq. (5), and a coordinate-descent-like update of u_i, as in Eq. (17). The key to this combination is the quantization of the external variable, arising from replacing the shrinkage operation with thresholding. As a result:

1. Due to quantization of the external variable, communication between nodes requires only low bandwidth and is kept to a minimum.
2. The choice of a component of $\boldsymbol{u}$ to be updated, in the sense of coordinate descent, is computed autonomously by each node.

To reduce bandwidth requirements, instead of communicating the analog variable $\boldsymbol{u}$, HDA nodes communicate a quantized variable $\boldsymbol{s} \in \{-1,0,1\}^n$ to each other. The variable $\boldsymbol{u}$, which solves (1) is obtained from $\boldsymbol{s}$ by averaging it over time: $\boldsymbol{u}=\lambda\bar{\boldsymbol{s}}=\frac{\lambda}{t}\sum_{k=0}^{t}\boldsymbol{s}^k$.

In HDA, components of $\boldsymbol{s}$ are obtained from the internal variable $\boldsymbol{\nu}$:

$$\boldsymbol{s} \leftarrow \text{threshold}(\nu, \lambda), \quad (18)$$

where threshold function is component wise,

$$\text{threshold}(x,\lambda)=\begin{cases} 1, & \text{if} x>\lambda \\ 0, & \text{if} -\lambda \le x \le \lambda \\ -1, & \text{if} x<-\lambda \end{cases}.$$

An update for the internal variable $\boldsymbol{\nu}$ is similar to (5) but with substitution of $\boldsymbol{u}$ by $\lambda\boldsymbol{s}$:

$$\nu \leftarrow \nu - \boldsymbol{A}^T(\lambda \boldsymbol{A}\boldsymbol{s} - \boldsymbol{f}). \quad (19)$$

Note that in HDA there is no need to explicitly specify the order in which the components of $\boldsymbol{u}$ are updated because the threshold operation (18) automatically updates the components in the order they reach threshold. Updates (18, 19) lead to the following computer algorithm.

Algorithm 2

Discrete-time HDA

Initialize: $\boldsymbol{\nu}=0$, $\boldsymbol{u}=0$, $\boldsymbol{s}=0$, $t=0$.

while "$\|\boldsymbol{f}-\boldsymbol{A}\boldsymbol{u}\|_2^2$ not converge" **do**

$t \leftarrow t+1$

$\boldsymbol{\nu} \leftarrow \boldsymbol{\nu} - \boldsymbol{A}^T(\lambda \boldsymbol{A}\boldsymbol{s} - \boldsymbol{f})$,

$\boldsymbol{s} \leftarrow$ threshold $(\boldsymbol{\nu}, \lambda)$,

$\boldsymbol{u} \leftarrow ((t-1)\boldsymbol{u} + \lambda \boldsymbol{s})/t$.

end while

Although not necessary, precomputing $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{f}$ may speed up algorithm execution.

To gain some intuition for Algorithm 2 consider an example, where $\boldsymbol{f}$ is chosen to coincide with some column of $\boldsymbol{A}$, i.e. $\boldsymbol{f}=\boldsymbol{A}_i$. Then the solution of problem (1) must be $u_i=1$, $u_{j\neq i}=0$. Now, let us see how the algorithm computes this solution.

The algorithm starts with $\boldsymbol{\nu}=0$, $\boldsymbol{u}=0$, $\boldsymbol{s}=0$. Initially, each component ν_j changes at a rate of $\boldsymbol{A}_j^T\boldsymbol{A}_i$ and, while the i-th component is below the threshold, $\boldsymbol{u}$ stays at 0. Assuming $\lambda \gg 1$, after $\lambda/(\boldsymbol{A}_i^T\boldsymbol{A}_i)=\lambda$ iterations, ν_i reaches the threshold λ and s_i switches from 0 to 1. At that time, the other components of $\boldsymbol{\nu}$ are still below threshold, $|\nu_{j\neq i}|=|\lambda \boldsymbol{A}_{j\neq i}^T f|=|\lambda \boldsymbol{A}_{j\neq i}^T \boldsymbol{A}_i|<\lambda$ and, therefore the components $s_{j\neq i}$ stay at 0. Note that choosing large λ guarantees that no more than one component reaches the threshold at any iteration.

Knowing $\boldsymbol{s}$, we can compute the next iteration for $\boldsymbol{\nu}$ (19), which is $\boldsymbol{\nu} = \lambda \boldsymbol{A}^T\boldsymbol{f} - \boldsymbol{A}^T(\lambda \boldsymbol{A}_i s_i - \boldsymbol{f})$ $= \lambda \boldsymbol{A}^T\boldsymbol{A}_i - \boldsymbol{A}^T\lambda \boldsymbol{A}_i + \boldsymbol{A}^T\boldsymbol{f} = \boldsymbol{A}^T\boldsymbol{f}$. Note that the first and the second terms cancelled because the change in ν accumulated over previous λ iterations is canceled by receiving broadcast s_i. Because s_i switches back to 0, $u_i=\lambda s_i=\lambda/\lambda=1$ as required. From this point on, the above sequence repeats itself. The above cancellation maintains $s_{j\neq i}=0$ and ensures sparsity of the solution, $u_{j\neq i}=0$.

The HDA updates (18, 19) can be immediately translated into the continuous-time evolution of the physical variables $\boldsymbol{s}(t)$ and $\boldsymbol{\nu}(t)$ in a hardware implementation.

Continuous-time evolution:

$$\nu(t)=\int_0^t \boldsymbol{A}^T[\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \quad (20)$$

$$\boldsymbol{s}(t)=\mathrm{spike}(\nu(t),\lambda), \quad (21)$$

where the spike function is component wise, $$\mathrm{spike}(\nu_i(t),\lambda)=\begin{cases} \delta_t, & \text{if} \nu_i(t)=\lambda \\ 0, & \text{if} -\lambda<\nu_i(t)<\lambda \\ -\delta_t, & \text{if} \nu_i(t)=-\lambda \end{cases}$$ and δ_t stands for a Dirac delta function centered at time t.

In this continuous-time evolution, the solution to (1) is given by the scaled temporal average $\boldsymbol{u}(t)=\frac{\lambda}{t}\int_0^t \boldsymbol{s}(t')dt'$.

The HDA can be naturally implemented on a neuronal network, Fig 1. Unlike the LCA (Rozell et al., 2008) and the LBI (Yin et al., 2008, Cai et al., 2009a, b), which require neurons continuously communicating graded potentials, the HDA uses perfect, or non-leaky, integrate-and-fire neurons (Koch, 1999, Dayan and Abbott, 2001). Ideal, or non-leaky, integrate-and-fire neurons integrate inputs over time in their membrane voltage, $\boldsymbol{\nu}$, (20) and fire a unitary action potential (or spike) when the membrane voltage reaches the threshold,

λ, (21). The inputs come from the stimulus, $\boldsymbol{A}^T\boldsymbol{f}$, and from other neurons, via the off-diagonal elements of $-\boldsymbol{A}^T\boldsymbol{A}$. After the spike is emitted, the membrane voltage is reset to zero due to the unitary diagonal elements of $\boldsymbol{A}^T\boldsymbol{A}$. We emphasize that, in discrete-time simulations, the membrane potential of HDA integrate-and-fire neurons after spiking is reset by subtracting the threshold magnitude rather than by setting it to zero (Brette et al., 2007).

Unlike thresholding in the HDA nodes (21), in biological neurons, thresholding is one-sided (Koch, 1999, Dayan and Abbott, 2001). Such discrepancy is easily resolved by substituting each node with two opposing (on- and off-) nodes. In fact, neurons in some brain areas are known to come in two types (on- and off-) (Masland, 2001).

Therefore, the HDA can be used as a model of computation with integrate-and-fire neurons. In the next section, we prove that $\boldsymbol{u}$, a time-average of $\boldsymbol{s}$, which can be viewed as a firing rate, converges to a solution of $\boldsymbol{f}=\boldsymbol{A}\boldsymbol{u}$.

Finally, for the sake of completeness, we propose the following "hopping" version of the HDA, which does not reduce energy consumption of communication bandwidth, yet is convenient for fast implementation on the CPU architecture for the sake of modeling.

Algorithm 3

hopping HDA

Initialize: $\boldsymbol{\nu}=0$, $\boldsymbol{u}=0$, $\boldsymbol{s}=0$, $t=0$.

While "$\|\boldsymbol{f}-\boldsymbol{A}\boldsymbol{u}\|_2^2$ not converge" **do**

$r=\max|\nu_i|$,

$j=\text{argmax}_i|\nu_i|$,

if $r<\lambda$ **then**

$$t_w=\min[(\lambda\text{sign}(\boldsymbol{A}_i^T\boldsymbol{f})-\nu_i)/(\boldsymbol{A}_i^T\boldsymbol{f})],$$

$$j=\text{argmin}_i[(\lambda\text{sign}(\boldsymbol{A}_i^T\boldsymbol{f})-\nu_i)/(\boldsymbol{A}_i^T\boldsymbol{f})],$$

$t\leftarrow t+t_w$,

$$s_j\leftarrow\text{sign}(\boldsymbol{A}_j^T\boldsymbol{f})$$

$\boldsymbol{\nu}\leftarrow\boldsymbol{\nu}+t_w\boldsymbol{A}^T\boldsymbol{f}-\lambda s_j\boldsymbol{A}^T\boldsymbol{A}_j$,

$u_j\leftarrow((t-1)u_j+s_j)/t$,

else

$s_j\leftarrow\text{sign}(\nu_j)$

$\boldsymbol{\nu}\leftarrow\boldsymbol{\nu}-\lambda\boldsymbol{A}^T\boldsymbol{A}_j s_j$,

$u_j\leftarrow((t-1)u_j+\lambda s_j)/t$,

end if

end while

As before, precomputing $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{f}$ may speed up algorithm execution.

The name "hopping HDA" comes from the fact that, instead of waiting for many iterations to reach the threshold, λ, the algorithm directly determines the component of $\boldsymbol{\nu}$ which will be the next to reach the threshold and computes the required integration time in t_w. Thus, the idea of hopping is similar to the ideas behind LARS (Efron et al., 2004) and "kicking"

(Osher et al., 2010). When that component of ν reaches the threshold, it broadcasts $-\boldsymbol{A}^T\boldsymbol{A}$ to other neurons instantaneously. We note that in practice, several nodes may exceed the threshold at the same time. In this case, we update super-threshold components based on the magnitude of ν_i starting with the largest.

5 Asymptotic performance guarantees

In this section, we analyze the asymptotic performance of the HDA by proving three theorems. Theorem 1 demonstrates that the HDA can be viewed as taking steps towards the solutions of a sequence of the Lasso problems whose regularizer coefficient decays in the course of iterations. Theorem 2 demonstrates that the representation error decays as *1/t* in the asymptotic limit. Theorem 3 demonstrates that, in the presence of time-varying noise, the representation error in the asymptotic limit decays also as a power of *t*. All the results are proven for the evolution described by Eqns. (20, 21), but can be easily adapted for the discrete-time case.

Importantly, Theorems 1 and 2 together suggest an intuition for why HDA finds a sparse solution. As the solution of a Lasso problem is known to be sparse (Tibshirani, 1996), it may seem possible that solving a sequence of the Lasso problems, as shown in Theorem 1, would yield a sparse solution. Yet, one may argue that, according to Theorem 1, the regularizer coefficient decays in the course of iterations and, because smaller regularization coefficients should yield less sparse solutions, the final outcome may not be sparse. Note, however, that the driving force for the growth of components of $\boldsymbol{u}$ is given by the representation error, which itself shrinks in the course of iterations according to Theorem 2. Because the error decays with the same asymptotic rate as the regularization coefficient we may still expect that the ultimate solution remains sparse. Indeed, such intuition is born out by numerical simulations as will be demonstrated in Section 6.

Theorem 1: Define average external variable at time *t* as $\bar{\boldsymbol{s}}(t) := \frac{1}{t}\int_0^t \boldsymbol{s}(t')dt'$. Then, provided $|\boldsymbol{s}(t)|_1 \neq 0$, the energy function $E(t) := \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 + (\lambda^2/t)\|\bar{\boldsymbol{s}}(t)\|_1$ generated by (20,21) decreases monotonically.

Proof: To prove this theorem, we consider separately the change in $E(t)$ during the interval between spikes and the change in $E(t)$ during a spike. We define $\boldsymbol{w} := \int_0^t \boldsymbol{s}(t')dt'$, which does not change during the interval between spikes. Then we replace $\boldsymbol{s}(t)$ in $E(t)$ by $\boldsymbol{w}/t$ and obtain after simple algebra:

$$dE(t)/dt = \frac{2\lambda}{t^3}\boldsymbol{w}^T\boldsymbol{A}^T(\boldsymbol{f}t - \lambda\boldsymbol{A}\boldsymbol{w}) - \frac{2\lambda^2}{t^3}\|\boldsymbol{w}\|_1 = \frac{2\lambda}{t^3}[\boldsymbol{w}^T\nu(t) - \lambda\|\boldsymbol{w}\|_1] = \frac{2\lambda}{t^3}\sum_{i=1}^n [w_i\nu(t)_i - \lambda|w_i|]. \quad (22)$$

The second equality follows from Eq. (20). Since $|\nu(t)_i| < \lambda$, if $|\boldsymbol{w}|_1 \neq 0$, $dE(t)/dt < 0$. Therefore, during the interval between spikes, $E(t)$ decreases.

If the *i*-th neuron fires a spike at *t*, $|s(t)_i| = 1$ and $s(t)_{j\neq i} = 0$, then the difference between $E(t)$, just after the spike, and $E(t^-)$, just before the spike is given by (notation t^- means arbitrarily close to *t* from below),

$$
\begin{aligned}
&E(t) - E(t^-) \\
&\quad = \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 \\
&\quad + (\lambda^2/t)\|\bar{\boldsymbol{s}}(t)\|_1 \\
&\quad - \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t^-)\|_2^2 \\
&\quad - (\lambda^2/t)\|\bar{\boldsymbol{s}}(t^-)\|_1 \\
&\quad = \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t^-) - \lambda s(t)_i \boldsymbol{A}_i/t\|_2^2 \\
&\quad - \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t^-)\|_2^2 \\
&\quad + \frac{\lambda^2}{t}\Big(\sum\nolimits_{j\neq i}\Big|\bar{\mathrm{s}}(t^-)_j\Big| \\
&\qquad + |\bar{\mathrm{s}}(t)_i|\Big) \\
&\quad - \frac{\lambda^2}{t}\Big(\sum\nolimits_{j\neq i}\Big|\bar{\mathrm{s}}(t^-)_j\Big| + |\bar{\mathrm{s}}(t^-)_i|\Big) \\
&\quad = -\frac{2\lambda}{t} s(t)_i \boldsymbol{A}_i^T (f \\
&\qquad - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t^-)) \\
&\quad + \frac{\lambda^2}{t^2}\|\boldsymbol{A}_i\|_2^2 \|s(t)_i\|_2^2 \\
&\quad + \frac{\lambda^2}{t}\big(|\bar{\mathrm{s}}(t)_i| \\
&\qquad - |\bar{\mathrm{s}}(t^-)_i|\big) \\
&\quad = -\frac{2\lambda}{t^2} s(t)_i \nu(t^-)_i \\
&\quad + \frac{\lambda^2}{t^2}\|s(t)_i\|_2^2 \\
&\quad + \frac{\lambda^2}{t}\big(|\bar{\mathrm{s}}(t^-)_i \\
&\quad + s(t)_i/t| \\
&\qquad - |\bar{\mathrm{s}}(t^-)_i|\big). \\
&\quad = \frac{\lambda^2}{t^2}\big[\|s(t)_i\|_2^2 + |s_i^t| \mathrm{sign}(\bar{\mathrm{s}}(t^-)_i s(t)_i) - 2s(t)_i \nu(t^-)_i/\lambda\big]
\end{aligned}
\tag{23}
$$

In the above equation, we used the relation $\boldsymbol{s}(t) = \boldsymbol{s}(t^-) + \boldsymbol{s}(t)/t$, which can be written separately for each component as $s(t)_i = s(t^-)_i + s(t)_i/t$ and $s(t)_{j\neq i} = s(t^-)_{j\neq i}$ (because $s(t)_{j\neq i} = 0$). Since $s(t)_i \nu(t^-)_i \to \lambda$, $\|s(t)_i\|_2^2 = |s(t)_i| = 1$, $E(t) - E(t^-) \to 0$ when $\mathrm{sign}(s(t^-)_i s(t)_i) = 1$ and $E(t) - E(t^-) < 0$ when $\mathrm{sign}(s(t^-)_i s(t)_i) = -1$. Therefore, at spike time, $E(t)$ does not increase. Combining (22) and (23) concludes the proof.

Similarly, for the discrete-time HDA, Algorithm 2, it is easy to show that, for sufficiently large λ, if $\|\bar{\boldsymbol{s}}(t) := (1/t)\sum_{k=0}^{t-1} \boldsymbol{s}^k\|_1 \neq 0$, the sequence $\{E(t) := \|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 + (\lambda^2/t)\|\bar{\boldsymbol{s}}(t)\|_1\}$ generated by Algorithm 2 decreases monotonically.

Theorem 2 : There exists an upper bound on the representation error, $|\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t)|_2$, which decays as $O(1/t)$.

Proof: In the continuous-time evolution, $\nu(t)_i = \boldsymbol{A}_i^T \int_0^t [\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt'$. Because of the threshold operation, $|\nu(t)_i| \leq \lambda$ and, therefore,

$$\left| A_i^T \int_0^t [\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right| \leq \lambda. \quad (24)$$

Then, assuming that $\boldsymbol{A}$ has full row rank, $\left\| \int_0^t [\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right\|_2$ must be also bounded from above. Then, the representation error can be expressed as:

$$\|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 = \frac{1}{t^2} \left\| \int_0^t [\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right\|_2^2 \leq \frac{\text{const}}{t^2}. \quad (25)$$

Therefore, $\|\boldsymbol{f} - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2 \leq \frac{const}{t}$, which concludes the proof.

Similar proof can be given for the discrete-time HDA, although with a different constant.

Theorem 3 : Assume the signal $\boldsymbol{f}$ is subject to time varying noise, i.e. $\boldsymbol{f}(t) = \boldsymbol{f}^0 + \boldsymbol{\varepsilon}(t)$. If $\left\| \int_0^t \boldsymbol{\varepsilon}(t')dt' \right\|_2^2 = \mathrm{O}(t^{\alpha<1})$, then $\lim_{t\to\infty} |\boldsymbol{f}^0 - \lambda \boldsymbol{A}\boldsymbol{s}(\bar{t})|_2 = 0$ and there exist some upper bound of $|\boldsymbol{f}^0 - \lambda \boldsymbol{A}\boldsymbol{s}(\bar{t})|_2$, which decays as $t^{-\min(1,1-\alpha)}$.

Proof: Because of the threshold operation, $\boldsymbol{\nu}$ is bounded from above:

$$\begin{aligned} &\|\boldsymbol{f}^0 - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 \\ &= \frac{1}{t^2} \left\| \int_0^t [\boldsymbol{f}(t') - \boldsymbol{\varepsilon}(t') - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right\|_2^2 \\ &= \frac{1}{t^2} \left\| \int_0^t [\boldsymbol{f}(t') - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right\|_2^2 \\ &- \frac{2}{t^2} \left\langle \int_0^t \boldsymbol{\varepsilon}(t')dt', \int_0^t [\boldsymbol{f}(t') - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right\rangle \\ &+ \frac{2}{t^2} \left\| \int_0^t \boldsymbol{\varepsilon}(t')dt' \right\|_2^2. \end{aligned} \quad (26)$$

Using again the fact that $|\nu(t)_i| = \left| \boldsymbol{A}_i^T \int_0^t [\boldsymbol{f}(t') - \lambda \boldsymbol{A}\boldsymbol{s}(t')]dt' \right|$ is bounded from above, we obtain

$$\|\boldsymbol{f}^0 - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 \leq \frac{c^2}{t^2} - \frac{2c}{t^2} \left\| \int_0^t \boldsymbol{\varepsilon}(t')dt' \right\|_2 + \frac{2}{t^2} \left\| \int_0^t \boldsymbol{\varepsilon}(t')dt' \right\|_2^2 = O\left(t^{-\min(2,2-2\alpha)} \right). \quad (27)$$

This concludes the proof. Next, we consider several examples of noise.

In the case of $\boldsymbol{f}$ contaminated by the white noise, $\int_0^t \boldsymbol{\varepsilon}(t')dt' = O(\sqrt{t})$, and the representation error converges as $1/\sqrt{t}$.

In the case of static noise where $\boldsymbol{\varepsilon}(t) = \boldsymbol{\varepsilon}$, we obtain:

$$\|\boldsymbol{f}^0 - \lambda \boldsymbol{A}\bar{\boldsymbol{s}}(t)\|_2^2 \leq \frac{c^2}{t^2} - \frac{2c}{t}\|\boldsymbol{\varepsilon}\|_2 + 2\|\boldsymbol{\varepsilon}\|_2^2, \quad (28)$$

which can be used as a stopping criterion in a de-noising application to prevent over-fitting.

6 Numerical results

In this section, we report the results of numerical experiments. In the first experiment, we search for sparse representation (1) of synthesized data using HDA. The elements of the matrix $\boldsymbol{A} \in \mathbb{R}^{64\times128}$ are chosen from a normal distribution and column-normalized by dividing each element by the l_2 norm of its column. For the noiseless case, we construct vector $\boldsymbol{f}$ as $\boldsymbol{A}\boldsymbol{u}^0$, where $\boldsymbol{u}^0 \in \mathbb{R}^{128}$ is generated by randomly selecting $nz = 10$ locations for non-zero entries sampled from a flat distribution between −0.5 and 0.5. Then, we apply the discrete-time HDA (Algorithm 2) using the network (Fig. 1) with 128 nodes. We set the spiking threshold λ=10 and obtain a solution, $\boldsymbol{u} = \lambda\bar{\boldsymbol{s}}$, which is compared with $\boldsymbol{u}^0$, Fig. 2.

As hardware implementations of HDA or neural circuits must operate on the incoming signal $\boldsymbol{f}$ contaminated by noise, which varies during the iterative computation, we analyze the performance of HDA in the presence of noise. To model such a situation we add time varying Gaussian white noise to the original signal $\boldsymbol{f}^0 = \boldsymbol{A}\boldsymbol{u}^0$. On each iteration step, we set each component $f_i^k = f_i^0(1+0.5\varepsilon_i^k)$, where the noise ε_i^k is independently picked from a normal distribution, $N(0,1)$. We found that, despite such a low signal-to-noise ratio, the HDA yields $\boldsymbol{u}$, which is close to the original $\boldsymbol{u}^0$, Fig. 3a. The relative residual decays as $1/\sqrt{t}$, Fig. 4b, as expected from $\sum_{k=1}^{t}\varepsilon^k = o(\sqrt{t})$.

Next we explore the performance of HDA relative to that of the LBI for a wide range of parameters. We present the results as a function of two variables: system indeterminacy $\alpha = m/n$ and system sparsity $\beta = nz/n$ (Charles et al., 2011), Fig. 4. We pick $n = 200$ and vary (α, β) in the range between 0.1 and 0.9. For each pair (α, β), we calculate the corresponding (m, nz) and sample 50 different realizations of the over-complete dictionary $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and the sparse signal $\boldsymbol{u}^0 \in \mathbb{R}^{200}$ satisfying $|\boldsymbol{u}^0|_0 = nz$. We then use HDA and LBI to calculate the corresponding sparse solutions $\boldsymbol{u}_{HDA}$ and $\boldsymbol{u}_{LBI}$. We compare the solution of each algorithm to $\boldsymbol{u}^0$ and plot the relative mean square error $\|\boldsymbol{u}_{HDA/LBI} - \boldsymbol{u}_0\|_2^2/\|\boldsymbol{u}_0\|_2^2$ in Fig. 4a and b. When the system is sufficiently sparse (small β) and determinate (large α), upper left corners of Fig. 4a and b, the solution to the basis pursuit problem (1) is unique and $\boldsymbol{u}_0$ is perfectly recovered (Chen et al., 1998). Under such condition, the solution of HDA is essentially identical to that of LBI as demonstrated in Fig. 4c, which shows the relative mean square difference between the HDA and the LBI solutions $\|\boldsymbol{u}_{HDA} - \boldsymbol{u}_{LBI}\|_2^2/\|\boldsymbol{u}_{LBI}\|_2^2$. When β gets larger and α gets smaller, the recovery is poor for both algorithms because the predefined $\boldsymbol{u}_0$ is not necessarily the solution with minimum l_1-norm and the solution to (1) is not unique (Chen et al., 1998). Therefore the sparse solutions found by HDA and LBI can be very different as revealed by the large difference in the bottom right corner of Fig. 3c, but they still have near identical l_1-norms, Fig. 4d. We calculate the relative mean difference between the l_1-norms as abs($|\boldsymbol{u}_{LBI}|_1 - |\boldsymbol{u}_{HDA}|_1$)/$|\boldsymbol{u}_{LB}|_1$ and find that the difference averaged over all points in Fig. 4d is only 5×10^{-3}.

To demonstrate that HDA also serves as model of neural computation, we test it with biologically relevant inputs and dictionary. We use SPAMS (Mairal et al., 2010) to train a four times over complete dictionary with 1024 elements from 16×16 image patches randomly sampled form whitened natural images (Olshausen and Field, 1996). These image patches are further processed by subtracting the mean and normalizing contrast by setting

variance to unity. The resulting dictionary elements have spatial properties resembling those of V1 receptive fields, Fig. 5a, (Olshausen and Field, 1996). Then we create a test data set containing 1000 image patches prepared in the same fashion as training image patches. We decompose these image patches using HDA over the learned dictionary and record the mean l_1-arc length of the representation coefficients $|\boldsymbol{u}|_1$ at various stopping relative residual. As a comparison we also simulate the decompositions using LBI, LCA and RDA. We found that HDA achieves similar representation error – sparsity tradeoff, Fig. 5b and c.

7 Summary

In this paper, we propose an algorithm called HDA, which computes sparse redundant representation using a network of simple nodes communicating using punctuate spikes. Compared to the existing distributed algorithms such as LCA and RDA, the HDA has lower energy consumption and demands on the communication bandwidth. Also, HDA is robust to noise in the input signal. Therefore, HDA is a highly promising algorithm for hardware implementations for energy constrained applications.

We propose three implementations of the HDA: a discrete-time HDA (Algorithm 2), a continuous-time evolution of the physical variable in a hardware implementation, and a hopping HDA (Algorithm 3) for fast computation on a CPU architecture.

Finally, HDA operation combines analog and digital steps (Sarpeshkar, 1998) and is equivalent to a network of non-leaky integrate-and-fire neurons suggesting that it can be used as a model for neural computation.

References

Attwell D, Laughlin SB. An energy budget for signaling in the grey matter of the brain. J Cereb Blood Flow Metab. 2001; 21:1133–1145. [PubMed: 11598490]

Baraniuk RG. Compressive sensing. Ieee Signal Proc Mag. 2007; 24:118–124.

Bertsekas, DP. Convex optimization theory. Belmont, MA: Athena Scientific; 2009.

Boerlin M, Deneve S. Spike-Based Population Coding and Working Memory. Plos Comput Biol. 2011; 7

Boyd, S.; Vandenberghe, L. Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press; 2004.

Bregman LM. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Computational Mathematics and Mathematical Physics. 1967; 7:200–217.

Brette R, Rudolph M, Carnevale T, Hines M, Beeman D, Bower JM, Diesmann M, Morrison A, Goodman PH, Harris FC, Zirpe M, Natschlager T, Pecevski D, Ermentrout B, Djurfeldt M, Lansner A, Rochel O, Vieville T, Muller E, Davison AP, El Boustani S, Destexhe A. Simulation of networks of spiking neurons: A review of tools and strategies. J Comput Neurosci. 2007; 23:349–398. [PubMed: 17629781]

Cai JF, Osher S, Shen ZW. Convergence of the Linearized Bregman Iteration for L(1)-Norm Minimization. Mathematics of Computation. 2009a; 78:2127–2136.

Cai JF, Osher S, Shen ZW. Linearized Bregman Iterations for Compressed Sensing. Mathematics of Computation. 2009b; 78:1515–1536.

Charles AS, Garrigues P, Rozell CJ. Analog Sparse Approximation with Applications to Compressed Sensing. 2011 arXiv 11114118.

Chen SSB, Donoho DL, Saunders MA. Atomic decomposition by basis pursuit. Siam Journal on Scientific Computing. 1998; 20:33–61.

Chklovskii DB, Schikorski T, Stevens CF. Wiring optimization in cortical circuits. Neuron. 2002; 34:341–347. [PubMed: 11988166]

Dattorro, J. Convex Optimization & Euclidean Distance Geometry. Palo Alto, CA: Meboo Publishing; 2008.

Dayan, P.; Abbott, LF. Computational and Mathematical Modeling of Neural System. Cambridge, MA: MIT Press; 2001. Theoretical Neuroscience.

Deneve S, Boerlin M. Implementing Dynamical systems with spiking neurons. COSYNE. 2011

DeWeese MR, Wehr M, Zador AM. Binary spiking in auditory cortex. J Neurosci. 2003; 23:7940–7949. [PubMed: 12944525]

Efron B, Hastie T, Johnstone I, Tibshirani R. Least angle regression. Annals of Statistics. 2004; 32:407–451.

Elad M, Matalon B, Shtok J, Zibulevsky M. Wide-angle view at iterated shrinkage algorithms - art. no. 670102. P Soc Photo-Opt Ins. 2007; 6701:70102–70102.

Friedman J, Hastie T, Hofling H, Tibshirani R. Pathwise Coordinate Optimization. Annals of Applied Statistics. 2007; 1:302–332.

Gallant JL, Vinje WE. Sparse coding and decorrelation in primary visual cortex during natural vision. Science. 2000; 287:1273–1276. [PubMed: 10678835]

Genkin A, Lewis DD, Madigan D. Large-scale Bayesian logistic regression for text categorization. Technometrics. 2007; 49:291–304.

Kavukcuoglu K, Sermanet P, Boureau Y, Gregor K, Mathieu M, LeCun Y. Learning Convolutional Feature Hierachies for Visual Recognition. Advances in Neural Information Processing Systems. 2010

Koch, C. Biophysics of Computation. New York: Oxford University Press; 1999.

Laughlin SB, Sejnowski TJ. Communication in neuronal networks. Science. 2003; 301:1870–1874. [PubMed: 14512617]

Lennie P. The cost of cortical computation. Curr Biol. 2003; 13:493–497. [PubMed: 12646132]

Li YY, Osher S. COORDINATE DESCENT OPTIMIZATION FOR l(1) MINIMIZATION WITH APPLICATION TO COMPRESSED SENSING; A GREEDY ALGORITHM. Inverse Problems and Imaging. 2009; 3:487–503.

Mairal J, Bach F, Ponce J, Sapiro G. Online Learning for Matrix Factorization and Sparse Coding. J Mach Learn Res. 2010; 11:19–60.

Masland RH. The fundamental plan of the retina. Nat Neurosci. 2001; 4:877–886. [PubMed: 11528418]

Olshausen BA, Field DJ. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature. 1996; 381:607–609. [PubMed: 8637596]

Olshausen BA, Field DJ. Sparse coding of sensory inputs. Curr Opin Neurobiol. 2004; 14:481–487. [PubMed: 15321069]

Osher S, Mao Y, Dong B, Yin WT. Fast Linearized Bregman Iteration for Compressive Sensing and Sparse Denoising. Commun Math Sci. 2010; 8:93–111.

Pearl J. Embracing Causality in Default Reasoning. Artificial Intelligence. 1988; 35:259–271.

Rozell CJ, Johnson DH, Baraniuk RG, Olshausen BA. Sparse coding via thresholding and local competition in neural circuits. Neural Comput. 2008; 20:2526–2563. [PubMed: 18439138]

Sarpeshkar R. Analog versus digital: extrapolating from electronics to neurobiology. Neural Comput. 1998; 10:1601–1638. [PubMed: 9744889]

Shapero S, Brüderle D, Hasler P, Rozell C. Sparse approximation on a network of locally competitive integrate and fire neurons. Cosyne. 2011

Tibshirani R. Regression shrinkage and selection via the Lasso. Journal of the Royal Statistical Society Series B-Methodological. 1996; 58:267–288.

Tubaishat M, Madria S. Sensor Networks: An Overview. IEEE Potentials. 2003; 22

Xiao L. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. J Mach Learn Res. 2010; 11:2543–2596.

Yin WT, Osher S, Goldfarb D, Darbon J. Bregman Iterative Algorithms for l(1)-Minimization with Applications to Compressed Sensing. Siam Journal on Imaging Sciences. 2008; 1:143–168.

Zou H, Hastie T. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society Series B-Statistical Methodology. 2005; 67:301–320.

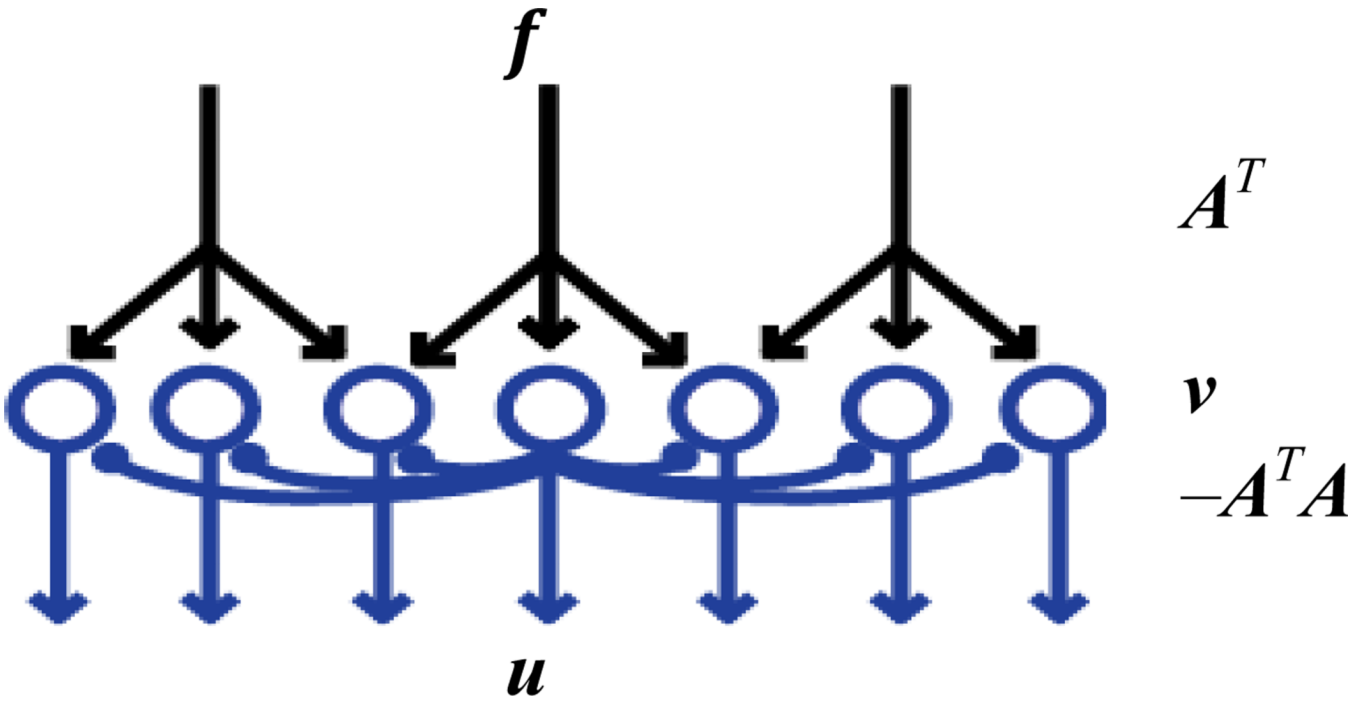

Figure 1.
A network architecture for LCA, RDA, LBI, or HDA. Feedforward projections multiply the input $\boldsymbol{f}$ by a matrix $\boldsymbol{A}^T$, while lateral connections update internal node activity $\boldsymbol{v}$ by a product of matrix $-\boldsymbol{A}^T\boldsymbol{A}$ and external activity $\boldsymbol{u}$.

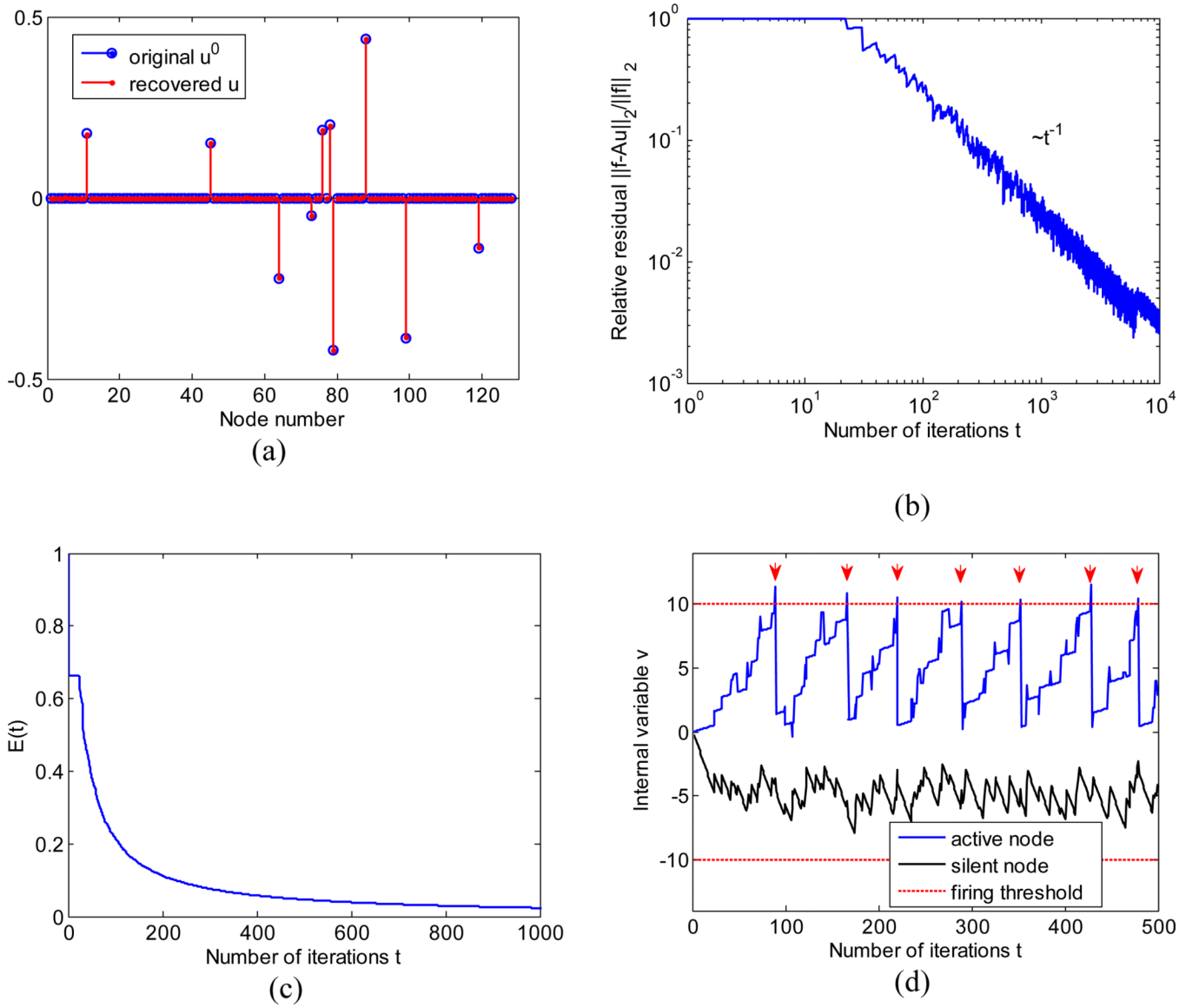

Figure 2.

Computing sparse representation, $\boldsymbol{u}$, from noiseless $\boldsymbol{f} = \boldsymbol{A}\boldsymbol{u}^0$ using HDA. (a) The reconstructed $\boldsymbol{u} = \lambda \boldsymbol{s}$ (stemmed red dots) at $t = 10000$ coincides with the original $\boldsymbol{u}^0$ (blue circles). (b) The relative residual $|\boldsymbol{f} - \lambda \boldsymbol{A}\boldsymbol{s}|_2 / |\boldsymbol{f}|_2$ decays as $1/t$ (note log-scale axes) in agreement with the upper bound (Theorem 2). The wiggles are due to the discreteness of $\boldsymbol{s}$. (c) Energy, E^t, as defined in Theorem 5.1 decays monotonically. (d) Representative evolution of internal variable, $\boldsymbol{v}$, of a broadcasting node (blue) and a silent node (black). Red arrows indicate time points when the component of $\boldsymbol{s}$ corresponding to the broadcasting node is non-zero. The firing thresholds (for λ=10) are shown by dashed red lines.

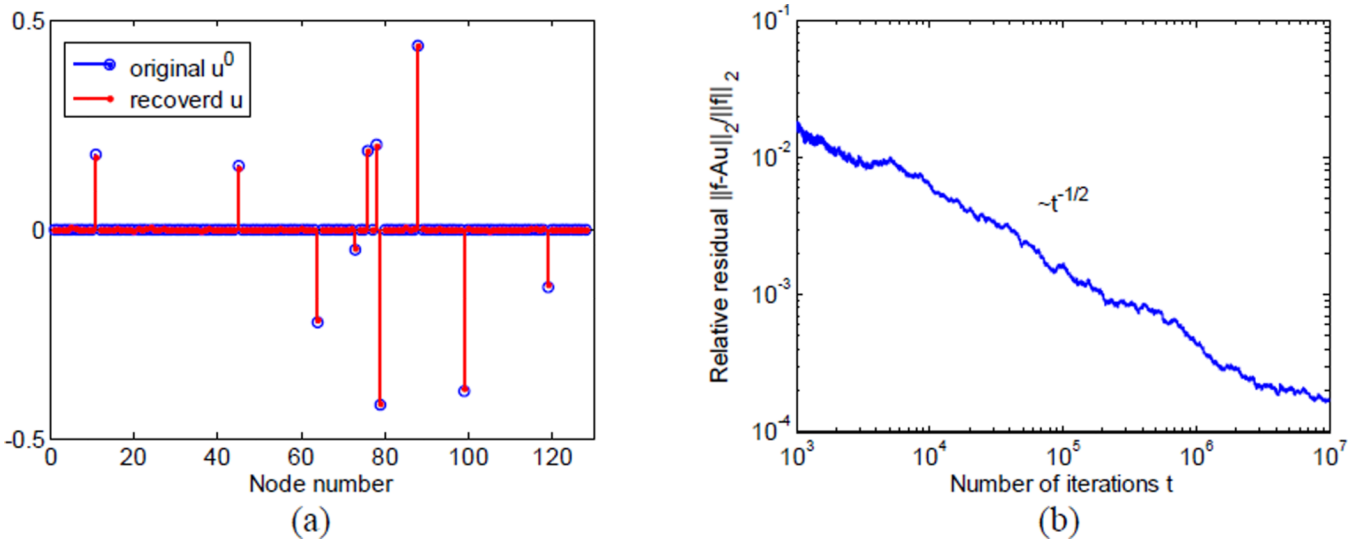

Figure 3.
The HDA is robust to noise in the input. Computing sparse representation on the same dataset as Figure 2 but contaminated by strong time-varying noise.

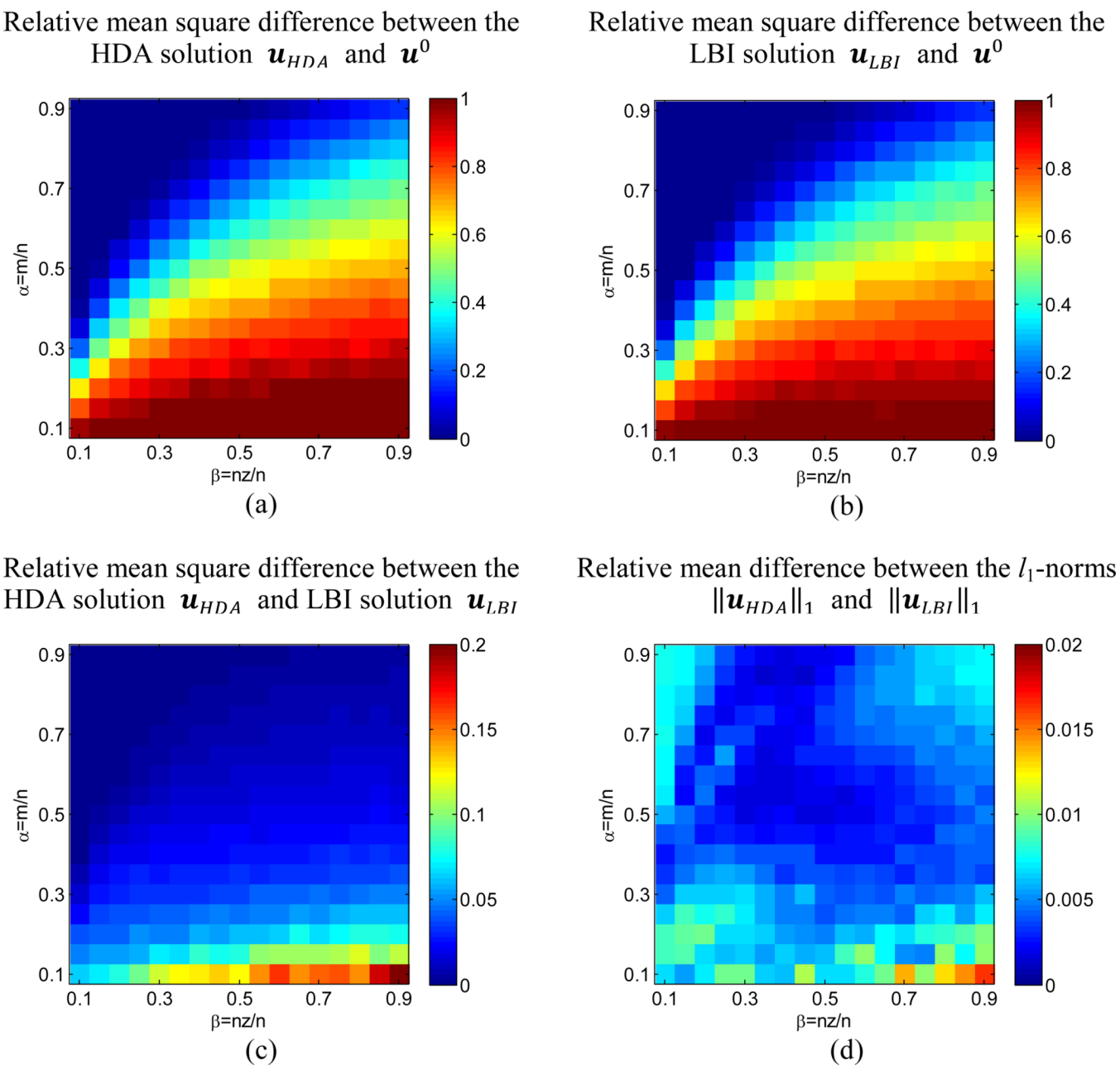

Figure 4.

For a wide range of parameters the HDA solution, $\boldsymbol{u}_{HDA}$, is on par with that of the LBI, $\boldsymbol{u}_{LBI}$. The relative mean square difference between $\boldsymbol{u}_{HDA}$ and the predefined sparse signal $\boldsymbol{u}^0$ (a) and the relative mean square difference between $\boldsymbol{u}_{LBI}$ and $\boldsymbol{u}_0$ (b) demonstrate both HDA and LBI both find the unique solution to the basis pursuit problem (1) when it exists (upper left corner). Indeed, the solutions $\boldsymbol{u}_{HDA}$ and $\boldsymbol{u}_{LBI}$ are essentially identical (c) and have the same l_1-norms $|\boldsymbol{u}_{HDA}|_1$ and $|\boldsymbol{u}_{LBI}|_1$ (d).

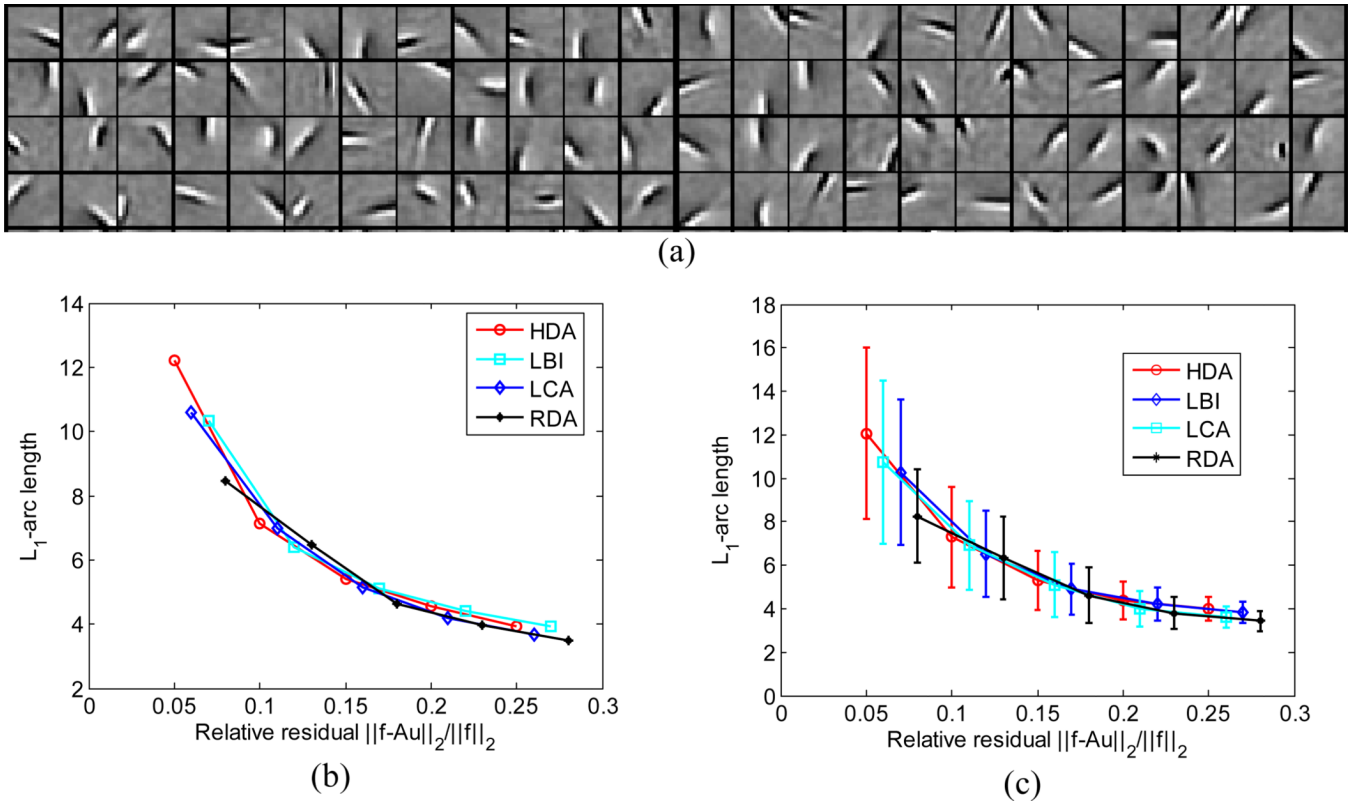

Figure 5.

HDA achieves error – sparsity tradeoff comparable with LBI, LCA and RDA. (a) Representative dictionary elements learned from whitened natural image patches. (b) Tradeoff for a typical natural image patch and (c) Mean tradeoff for an ensemble of 1000 contrast normalized image patches.