Toward a Robust Sparse Data Representation for Wireless Sensor Networks

Mohammad Abu Alsheikh*†, Shaowei Lin†, Hwee-Pink Tan‡, and Dusit Niyato*
*School of Computer Engineering, Nanyang Technological University, Singapore 639798
†Sense and Sense-abilities Programme, Institute for Infocomm Research, Singapore 138632
‡School of Information Systems, Singapore Management University, Singapore 188065

***Abstract*—Compressive sensing has been successfully used for optimized operations in wireless sensor networks. However, raw data collected by sensors may be neither originally sparse nor easily transformed into a sparse data representation. This paper addresses the problem of transforming source data collected by sensor nodes into a sparse representation with a few nonzero elements. Our contributions that address three major issues include: 1) an effective method that extracts population sparsity of the data, 2) a sparsity ratio guarantee scheme, and 3) a customized learning algorithm of the sparsifying dictionary. We introduce an unsupervised neural network to extract an intrinsic sparse coding of the data. The sparse codes are generated at the activation of the hidden layer using a sparsity nomination constraint and a shrinking mechanism. Our analysis using real data samples shows that the proposed method outperforms conventional sparsity-inducing methods.**

***Abstract*—Sparse coding, compressive sensing, sparse autoencoders, wireless sensor networks.**

I. Introduction

A sparsely-activated data (a few nonzero elements in a sample vector) may naturally exist for compressive sensing (CS) applications in wireless sensor networks (WSNs) such as the path reconstruction problem [1], indoor localization [2], and sparse event detection [3]. On the other hand, a sparse data representation cannot be easily induced in many other real-world contexts (e.g., in meteorological applications and environmental data gathering). In particular, noise patterns are usually presented in collected data from WSNs which greatly affect the performance of conventional sparsity-inducing (transformation) algorithms such as the Haar wavelet and discrete cosine transforms [4]. This motivates the quest for noise-robust and effective sparsity-inducing methods for WSNs.

One of the breakthroughs in recent deep learning paradigms for finding high level data abstractions is achieved by introducing sparsity constraints on data representations, e.g., the Kullback–Leibler divergence [5], rectifier function [6], and topographic coding [7]. These methods are introduced for extracting intrinsic features from the data in a similar way that the human brain does while encoding sensory organ data, e.g., the low percentage of spikes in a visual cortex [8]. In particular, sparse deep learning methods generate sparse representations across training data for each single unit (i.e., *lifetime sparsity*), and they neither guarantee sparsity for each input signal nor assert on the number of nonzero values in the sparse codes. However, a practical CS implementation in WSNs requires a sparse representation for each input signal (i.e., *population sparsity*) with a sparsity ratio guarantee. Specifically, the CS solution to the underdetermined system (more number of unknowns than the number of equations) is dependent on the sparsity ratio of the signal, and the sparsity-inducing mechanism must assert an upper limit for the sparsity ratio. This sparsity bounding is necessary in WSNs as it enables using only one flat acquisition matrix for data encoding in the node. Therefore, it reduces the CS overhead in terms of memory for storing many measurement matrices in transmitting node and data control exchange as there is no need to send out rate control messages.

The main contributions of this paper can be summarized into three folds as follows.

1) This paper introduces an effective, population sparsity-inducing algorithm with sparsity ratio guarantee. The algorithm is based on a customized unsupervised neural network model of three layers (also called an autoencoder network) that generates the required, sparse coding at the second (hidden) layer. In the proposed *shrinking sparse autoencoder* (SSAE), the sparsity is achieved by introducing a regularization term to the cost function of the basic autoencoder.
2) We customize the learning algorithm to meet WSN characteristics. For example, the activations of the hidden layer during parameter learning stage are rounded to only three places to consider limited computational precision of the node. The rounding considers the low precision computations of sensor nodes, and it reduces the compressed data size and data transmission load.
3) We present a customized learning method that optimizes the SSAE cost function. Basically, the back propagated error is only used to update the nonzero and active neurons with dominant output values for each input pattern. Moreover, a shrinking mechanism that guarantees the sparsity bound is also used during the learning of the SSAE's parameters. Therefore, an SSAE asserts on the number of nonzero elements generated at any time instant.

The literature is rich with sparsity-based methods that are designed for optimized WSN operations [1]–[4], [9]–[14]. Nonetheless, much less attention is given to the sparsity-inducing stage, and using straightforward methods to extract

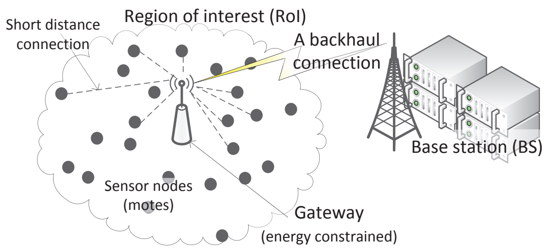


Fig. 1: Compressive sensing (CS) based data aggregation model: The RoI is assumed to be relatively far from the BS. Therefore, a gateway is designed to transmit compressed data over a costly long distance wireless connection.

sparsity basis is common in previous studies such as using principal component analysis (PCA) [10], discrete cosine transform (DCT) [4], [11], [12], discrete Fourier transform (DFT) [9], discrete wavelet transforms [4], [13], and difference matrices [4], [15]. However, the sparse coding discipline has evolved considerable advances that significantly enhance the sparsity-inducing and hence overall WSN operations. Therefore, this paper is intended to introduce a robust and more effective sparsity-inducing method. The proposed method consists of three steps: (i) data collection, (ii) offline training and modeling, and (iii) online sparse code generation. An example of the online sparse code generation for a CS application is shown in Figure 2 which will be described in details later.

The rest of the paper is organized as follows. In Section II, the problem formulation is presented. Section III describes the proposed algorithm and the SSAE structure. Then, Section IV discusses important practical issues of training and fitting the proposed model. In Section V, numerical results using real-world data set are presented. Finally, Section VI summarizes this paper.

II. Problem Formulation

Consider a dense wireless sensor network consisting of N nodes, as in Figure 1, that collects data about a region of interest (RoI). Each sensor i (where $i = 1, \dots, N$) collects a real-valued sample x_i (e.g., temperature measurements) at a predefined sampling period and transmits packets at a configured transmission power that is not sufficient to reach the base station (BS) due to long distance propagation. Therefore, a gateway (GW) is used to collect a data vector $\mathbf{x} \in \mathbb{R}^N$ from all sensor nodes and relay it to the BS for further analysis and processing. Thereafter, a historical data matrix $\mathbf{X} \in \mathbb{R}^{T\times N}$ is formulated at the BS containing the collected data vectors as its rows, where T is the number of collected vectors. Here, the sensors' oscillators are assumed to be synchronized to the GW's clock.

After collecting sufficient historical samples (details of data collection are elucidated in Section IV-A), and as the GW is energy and bandwidth constrained, the GW employs CS to spatially compress the data into a smaller data size. The radio transceiver is the most energy consuming unit in an ordinary sensor node [16]. Thereby, the energy consumption becomes more critical in the GW unit as it transmits huge data over the backhaul connection, while sensor nodes are assumed to transmit for short distances. It is important to note that our algorithm can be also temporally applied at each individual sensor node. However, data delivery latency is provoked as temporal samples must be collected at the node before being transmitted as one compressed chunk. Next, we give an overview of the CS framework and its implementation at the GW device, and the data reconstruction at the BS unit.

A. Compressive Sensing (CS)

CS is a signal processing method for effective data recovery from a few data samples than the Nyquist rate [17]. Assuming a sparse signal $\mathbf{s} \in \mathbb{R}^L$ that has only K nonzero elements; therefore, $\mathbf{s}$ is called a K-sparse signal, and the *sparsity ratio* η is equal to $\frac{K}{L}$. Moreover, suppose a measurement (or sensing) matrix $\mathbf{\Phi} \in \mathbb{R}^{M\times L}$ that obeys the restricted isometry property (RIP) [18]. Here, M is assumed to be much smaller than L; therefore, $\mathbf{\Phi}$ is a flat matrix with more columns than rows. The sensing system under consideration that is executed by the GW to compress data can be expressed as

$$\mathbf{y} = \mathbf{\Phi}\mathbf{s} \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^{\mathbf{M}}$ is the resulted measurement vector. $\mathbf{\Phi}$ can be sampled from different distributions to meet the RIP such as the Gaussian distribution [19]. Moreover, for high probability recovery, M must also meet the following constraint [20]:

$$M \geq \rho K \log_2\left(\frac{L}{K}\right) \tag{2}$$

where ρ is a constant, and $M \ll N$. At the BS unit, the reconstruction of $\mathbf{s}$ from $\mathbf{y}$ can be achieved by minimizing the following relaxed problem [21]:

$$\mathbf{s}^* = \arg \min_{\|\mathbf{y}-\mathbf{\Phi}\mathbf{s}\|_2 \leq \epsilon} \|\mathbf{s}\|_1 \tag{3}$$

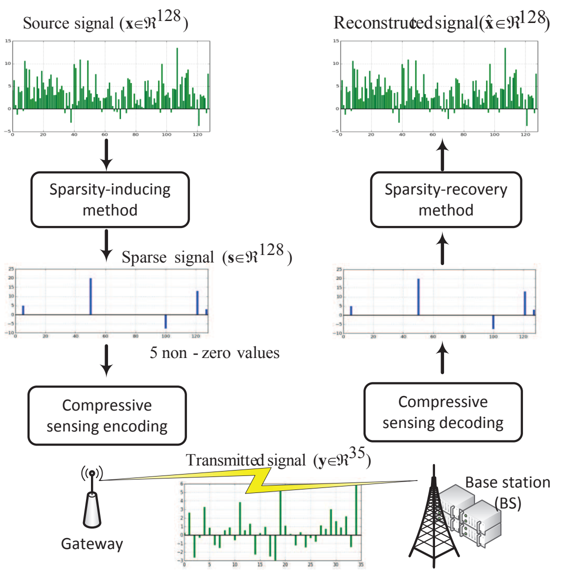
where ϵ is a small constant. The optimization problem (3) can be solved using a regularized least square method called least absolute shrinking and selection operator (LASSO) [22].

B. Sparsity-inducing

Clearly, the whole CS framework is based on the sparsity assumption. Natural signal such as sound and images can be transformed into a sparse form by projecting them into a suitable basis [17]. However, this is not the case when dealing with WSN data. More precisely, sensor nodes produce noisy readings of the form

$$\mathbf{x} = \mathbf{x}^* + \mathbf{z} \tag{4}$$

where $\mathbf{x}^* \in \mathbb{R}^N$ is the noiseless data vector of the physical phenomenon, and $\mathbf{z} \in \mathbb{R}^N$ is the added noise vector. Noise values are assumed to be independent Gaussian variables with zero mean and variance σ_z^2 such that $z \sim \mathbb{N}\left(0, \sigma_z^2 I_N\right)$. Therefore, even through the neighbor sensors are spatially correlated and hence compressible, the noise existence hampers the accurate approximation of source signal $\mathbf{x}$ using linear projection methods. In particular, smooth signal are representable using linear combinations of Fourier bases, and



Source signal ($\mathbf{x}\in\Re^{128}$)
Reconstructed signal ($\hat{\mathbf{x}}\in\Re^{128}$)
Sparsity-inducing method
Sparsity-recovery method
Sparse signal ($\mathbf{s}\in\Re^{128}$)
5 non - zero values
Compressive sensing encoding
Compressive sensing decoding
Transmitted signal ($\mathbf{y}\in\Re^{35}$)
Gateway
Base station (BS)

Fig. 2: Example of data compression, transmission, and recovery operations using CS and sparsity-inducing models.

smooth piecewise signals are linearly representable in wavelet bases [11]. Nonetheless, the smoothness condition is not guaranteed in sensor data as data samples are usually affected by noise patterns, and commercial sensors sense phenomenon with finite precision and are not noise robust. For example, a few noise readings can destroy the sparsity pattern of a DCT transformed data [12].

The main aim of any robust sparsity-inducing mechanism is to transform the source signal $\mathbf{x} \in \mathbb{R}^N$ into the sparse signal $\mathbf{s} \in \mathbb{R}^L$. An upper bound guarantee on the sparsity ratio of the generated signal $\mathbf{s}$ is a "must-have" feature in most sparsity-based applications such as in CS. In particular, this guarantee enables designing low memory and low communication overhead applications for WSNs as a single sensing matrix $\mathbf{\Phi}$ is used by the GW unit to compress data. Then, the BS does not require any information from the GW to recover the reconstruction signal $\overline{\mathbf{x}}$ other than the measurement vector $\mathbf{y}$, where $\overline{\mathbf{x}}$ is a reconstruction of the noiseless data vector $\mathbf{x}^*$.

An example of the system online operational procedure is shown in Figure 2 which includes the sparsity-inducing and CS components. The next section presents the proposed sparsity-inducing mechanism.

III. Shrinking Sparse Autoencoder (SSAE)

In this section, we introduce an autoencoder's variant which we call shrinking sparse autoencoder (SSAE) as shown in Figure 3. The SSAE network is specially designed to transform sensory data from its original domain into a sparse domain. The SSAE structure consists of three neural (or computational unit) layers. Firstly, an input layer that is connected to the input signal $\mathbf{d} \in \mathbb{R}^N$, where N is the number of sensor nodes in the network. Briefly, $\mathbf{d}$ is a sphered version of the raw sensor data $\mathbf{x}$, where $\{d_i \in \mathbb{R} | -1 < d_i < 1\}$ as described in

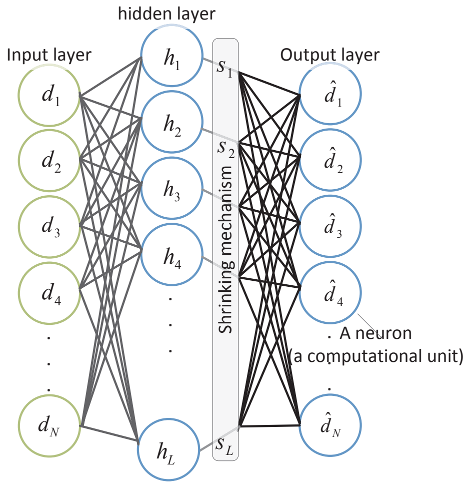


Fig. 3: Illustration of the SSAE structure.

Section IV-B. Secondly, a hidden layer is used to generate an intrinsic code $\mathbf{h} \in \mathbb{R}^L$ at its activation. Thirdly, an output layer that includes the same number of neurons as the input layer and generates a recovery of the input data $\hat{\mathbf{d}} \in \mathbb{R}^N$. The layers are connected to each other using the following formulations:

$$\mathbf{h} = f\left(\mathbf{W}^{(1)}\mathbf{d} + \mathbf{b}^{(1)}\right) \quad (5)$$

$$\hat{\mathbf{d}} = f\left(\mathbf{W}^{(2)}\mathbf{s} + \mathbf{b}^{(2)}\right) \quad (6)$$

where $\mathbf{W}^{(1)}$ is the weight matrix connecting the input and hidden layers, $\mathbf{W}^{(2)}$ is the weight matrix connecting the hidden and output layers, and $\mathbf{b}^{(1)}$ and $\mathbf{b}^{(2)}$ are the biases of the input and hidden layers, respectively. Additionally, $\mathbf{s}$ is the sparse data representation that is obtained by applying the shrinking operation as described in Section III-A. For simplicity, we define $\boldsymbol{\theta}$ to contain all the SSAE's parameters such that $\boldsymbol{\theta} \doteq \left[\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\right]$. Moreover, $f(\cdot)$ is the non-linear hyperbolic tangent function.

The SSAE's cost function $\Gamma(\cdot)$ includes two terms as follows:

$$\Gamma(\boldsymbol{\theta}; \mathbf{D}) = \frac{1}{T}\left(\sum_{u=1}^{T} \frac{1}{2}\left\|\widehat{\mathbf{d}}^{(u)} - \mathbf{d}^{(u)}\right\|^2\right) + \frac{\gamma}{T}\left(\sum_{u=1}^{T} \log_{10}\left(1 + \left(\mathbf{h}^{(u)}\right)^2\right)\right) \quad (7)$$

where $\mathbf{D} \in \mathbb{R}^{T \times N}$ is the training matrix of historical data such that each input vector $\left\{\mathbf{d}^{(u)}\right\}_{u=1}^{T}$ is stored in a row of this matrix, and $\mathbf{h}^{(u)}$ is the hidden layer activation of $\mathbf{d}^{(u)}$. Moreover, T is the training set size configured at the offline training algorithm (the details are given in Section III-B). As with any other autoencoder, the first term is the average sum of the difference between input vectors and their reconstructions at the output layer. This term is used to encourage the neural network to reconstruct its input data at the output layer. The second term is used to encourage sparsity at the generated coding in the hidden layer. The *sparsity penalty* γ is a hyper-parameter to manage the weights of each term in the optimization problem. In other words, using a big value for γ results in highly sparse representation, but with poor reconstruction capability. Then, the well-known delta rule can be used to update the SSAE's weights and biases as follows [23]:

$$W_{ij}^{(q)} \doteq W_{ij}^{(q)} - \alpha \frac{\partial}{\partial W_{ij}^{(q)}} \Gamma(\boldsymbol{\theta}; \mathbf{D}), \tag{8}$$

$$b_i^{(q)} \doteq b_i^{(q)} - \alpha \frac{\partial}{\partial b_i^{(q)}} \Gamma(\boldsymbol{\theta}; \mathbf{D}) \tag{9}$$

where α is the *learning rate*, and $q \in \{1, 2\}$ is the layer number within the SSAE network. These update rules are executed at each iteration of a gradient descent method. The partial derivative is given by

$$\frac{\partial}{\partial W_{ij}^{(q)}} \Gamma(\boldsymbol{\theta}; \mathbf{D}) = \frac{1}{T} \sum_{u=1}^{T} \frac{\partial}{\partial W_{ij}^{(q)}} \Gamma\left(\boldsymbol{\theta}; \mathbf{d}^{(u)}\right) \tag{10}$$

where $\Gamma\left(\boldsymbol{\theta}; \mathbf{d}^{(u)}\right)$ is the cost function defined for a single sample $\mathbf{d}^{(u)} \in \mathbf{D}$. This means that the overall partial derivative of (7) is found by averaging the partial derivatives of all input samples. The second term of (7) only affects the partial derivative of the hidden layer ($q = 2$) which is computed as follows:

$$\frac{\gamma}{T} \sum_{u=1}^{T} \frac{\partial}{\partial W_{ij}^{(2)}} \left(\gamma \log_{10} \left(1 + \left(\mathbf{h}^{(u)} \right)^2 \right) \right) =$$
$$\frac{\gamma}{\log_e(10) \times T} \sum_{u=1}^{T} \left(\frac{2\mathbf{h}^{(u)}}{1 + \left(\mathbf{h}^{(u)} \right)^2} \right) f'\left(\mathbf{W}^{(2)} \mathbf{d}^{(u)} + \mathbf{b}^{(2)} \right) \tag{11}$$

where $f'(\cdot) = 1 - (f(\cdot))^2$ is the element-wise derivative of $f(\cdot)$. Thereby, the SSAE is designed to generate many zeros at the hidden layer. One can think of a neuron as being active when its output is not equal to zero, and an inactive neuron does not participate in forwarding the input data to the output (because it does not generates signals). To this end, two important issues of the second term of (7) must be noted as follows:

- The second term minimizes the hidden layer activation, but it still does not ensure exactly zero activations.
- It does not guarantee sparsity ratio at the generated codes.

Accordingly, a shrinking mechanism must be applied at the hidden layer activation and before propagating them to the output layer to reconstruct the input. In particular, one can think of the second term as only being used as a mechanism of nominating the most promising neurons to be deactivated by the shrinking mechanism as described in the next section.

A. Shrinking (Pruning) Scheme

Even though the cost function of the SSAE is designed to generate a sparse data coding at the hidden layer, it does still neither guarantee a coding with population sparsity (sparsity at each input vector) nor assert on the maximum nonzeros for each input. Equally important, it will most likely generate values close to, but not absolutely zero. Therefore, we propose a simple shrinking mechanism that can complete the design cycle. For each input vector, the proposed shrinking mechanism "zero out" the least dominant neurons from the hidden layer, and therefore switching them to the deactivation mode. The least dominant neurons are the ones with the least effect on the data reconstruction at the output layer, and hence the minimum activation values that result from the sparsity restrictions. Therefore, only K active neurons at the hidden layer forward propagate the input through the SSAE network, and the remaining $L - K$ neurons are switched off. An optimized implementation of the shrinking scheme is given by the pseudo-code in Algorithm 1, where $|\cdot|$ is the absolute value function.

Algorithm 1 Pseudo-code for the shrinking operation of hidden layer's neurons.

```
1:  Input $\mathbf{h} \in \mathbb{R}^L$: hidden layer activation before shrinking
2:  Input $K$: maximum nonzero activations
3:  $\mathbf{s} = \mathbf{h}$                          ▷ copy operation
4:  for $i = 0$ to $L - K$ do
5:      $p = 0$
6:      for $j = 0$ to $(L - 1)$ do
7:          if $|s_p| > |s_j|$ and $|s_j| > 0$ then
8:              $p = j$
9:          end if
10:     end for
11:     $s_p = 0$                          ▷ zero-out smallest value
12: end for
13: Output $\mathbf{s} \in \mathbb{R}^L$
```

B. Offline Training

The SSAE's parameter adjustment is done during an offline training stage. As a resource demanding process, the training must be performed at the BS unit, and then the SSAE's parameters ($\boldsymbol{\theta}$) are disseminated for online data compression at GW. The learning stage and SSAE's parameters are tuned using a resourceful BS with relatively high precision operations. However, GW is usually constrained in terms of computational resources and computational precision (i.e., the machine epsilon value). Therefore, rounding the activation at the hidden layer is needed during the learning stage to match the GW's low precision. Moreover, with rounding, less data is transmitted from GW to BS.

To learn the SSAE's parameters ($\boldsymbol{\theta}$), we minimize (7) by using a non-linear quasi-Newton method called the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method [24]. However, firstly the collected historical data $\mathbf{X} \in \mathbb{R}^{T \times N}$ must be randomly shuffled. This is because sensors' readings are highly correlated over time, and a non-shuffled data causes the SSAE to dominantly learn the training data' patterns in training data only. Therefore, the shuffling step ensures that the training and testing data sets contain all possible data patterns. Moreover, the cross validation

Algorithm 2 The offline training algorithm.

```
1: Input $\mathbf{X} \in \mathbb{R}^{T \times N}$: historical sensor data
2: Input $K$: maximum nonzero activations
3: Input $\gamma$: sparsity hyper-parameter
4: Input $\varphi$: number of folds for cross validation
5: Randomly shuffle $\mathbf{X}$
6: Divide $\mathbf{X}$ into $\varphi$ folds $\mathbf{X}^1, \ldots, \mathbf{X}^\varphi$
7: for all $\mathbf{X}^i, i = 1, \ldots, \varphi$ do
8:     for all $\mathbf{x} \in (\mathbf{X} \setminus \mathbf{X}^i)$ do    $\triangleright$ held out $\mathbf{X}^i$ for testing
9:         Sphere input $\mathbf{x}$ to get $\mathbf{d}$ using (12)
10:        Append $\mathbf{d}$ to $\mathbf{D}$
11:    end for
12:    repeat
13:        for all $\mathbf{d} \in \mathbf{D}$ do
14:            Forwardly propagate $\mathbf{d}$ to compute $\mathbf{h}$ using (5)
15:            Shrink $\mathbf{h}$ to get $\mathbf{s}$ as in Algorithm 1
16:            Compute $\hat{\mathbf{d}}$ using (6)
17:        end for
18:        Compute the cost value using (7)
19:        Compute the gradient vector as in (10)
20:        Update $\boldsymbol{\theta}$ using the L-BFGS method
21:    until learning converges
22:    Compute accuracy using $\mathbf{X}^i$
23: end for
24: Compute average accuracy of the $\varphi$ folds
25: Output $\boldsymbol{\theta} \doteq \left[\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\right]$
```

technique [25] is an effective method for testing the model generalization capabilities, while benefiting from all available samples for training. Cross validation divides the training data into φ groups (e.g., 10 groups) then at each time, one group is held out for testing while using the remaining for model fitting. Then, the model performance is found by averaging errors of all cross validation's groups. The offline learning is described in Algorithm 2.

The learning algorithm is computationally intensive for sensor nodes and must be performed at the BS. Moreover, if the statistical parameters of the underlying phenomenon change, the offline training must be re-executed and an updated $\left[\mathbf{W}^{(1)}, \mathbf{b}^{(1)}\right]$ should be disseminated to the nodes.

C. Computational Complexity

The online encoding and decoding of sparse codes are lightweight. In particular, the GW (or a sensor node) can generate sparse codes by only using $\left[\mathbf{W}^{(1)}, \mathbf{b}^{(1)}\right]$ as in (5) and Algorithm 1 with $\mathcal{O}(L \times N)$ of overall time complexity. The data recovery is performed at the BS by using $\left[\mathbf{W}^{(2)}, \mathbf{b}^{(2)}\right]$ as in (6) with a similar time complexity of $\mathcal{O}(L \times N)$.

D. Sparse Codes

For the verification and analysis in the following sections, a meteorological data set from the Sensorscope project [26] is used. The data set contains surface temperature samples of 23 sensors. The learning curve of the SSAE is shown in Figure 4a.

An important indication of successful SSAE training is ensuring that hidden neurons are not connected with zero weights to the input layer. In other words, this ensures that any neuron in the hidden layer will be active for some input patterns, and hence no "always-off" neuron exists. This increases the model performance of generalizing to non local data, and hence it performs well on extremely non linear data, as all neurons participating increases the possible code formulations (i.e., the number of distinct combinations is increased when having more active neurons). Figure 4b shows hidden layer activations over time. Here, two main desirable properties can be observed

1) Population sparsity is achieved, and the maximum number of active neurons at any time instant is guaranteed by the SSAE network. This upper bound of nonzeros in a generated sparse code considers the tradeoff between the recovery error and compression ratio of the data aggregation model. Therefore, only a single sensing matrix is needed when using CS to create a measurement vector at the GW node.
2) All neurons are participating in the sparse code generation, and without any "always-off" neuron. Moreover, the activation values of the active neurons are not concentrated around very small values near zero. This feature cannot be achieved in conventional average activity ratio sparse autoencoders, such as the Kullback–Leibler divergence, as they are designed for lifetime sparsity only.

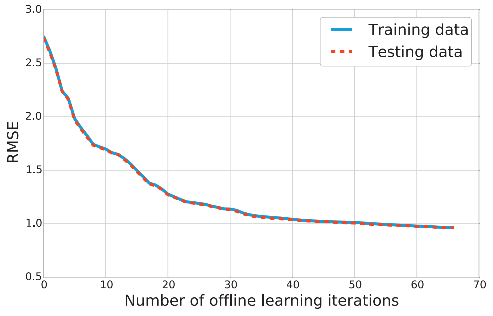
IV. Discussion and Practical Considerations

In this section, some practical issues of the SSAE training and fitting are discussed.

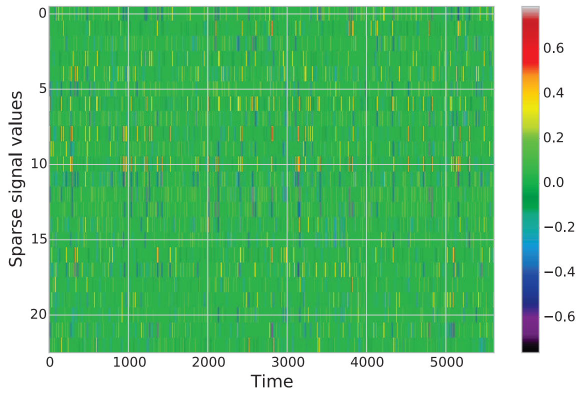
A. Data Collection

A crucial aspect of machine leaning-based approaches, such as the SSAE network, is the training data requirement. A system designer may have access to a large historical data set that is collected in the past. This historical data can be used to train the SSAE's model. However, this is not the case for new WSN's deployments, and the lack of sensor data hinders the accurate fitting of the SSAE's parameters (i.e., $\boldsymbol{\theta}$). Clearly, the SSAE's model needs to globally generalize to unseen data samples. In any machine learning method, having more training data can improve generalization performance, but having more data is not the only solution [27]. In WSNs, the following issues must be considered when using an SSAE as a sparsity inducing method.

1) It is assumed that sensor nodes are densely deployed and hence spatially correlated with each other (e.g., as in Figure 5 for the Sensorscope project's data). SSAE learns these spatial correlation and redundant patterns in the nodes' collected data. Therefore, if the underlying phenomenon becomes different in the way that it changes the nodes' spatial correlation, then new data collection and offline model fitting must be performed.



(a)



(b)

Fig. 4: An SSAE which is designed to produce a maximum of 5 nonzero values at each time instant ($\eta = 0.2$) for 23 sensors (i.e., $\mathbf{x} \in \mathbb{R}^{23}$). (a) A learning curve that shows the convergence of the offline learning algorithm, and (b) activation values of the hidden layer's neurons.

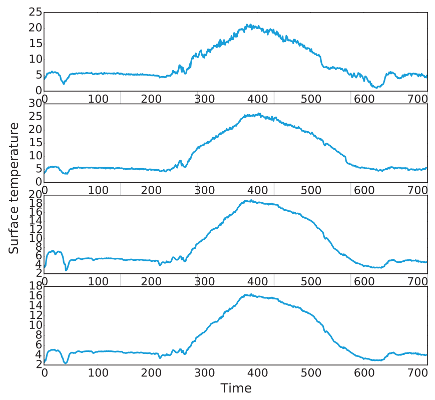
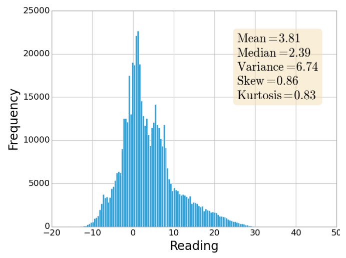
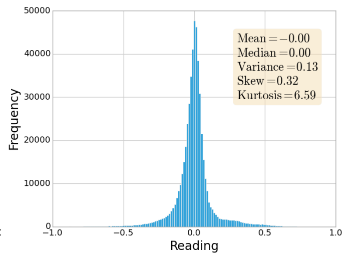


Fig. 5: Surface temperature readings of 4 neighbor sensors from the Sensorscope deployment over 1 day (1 sample every 2 minutes). This shows the spatial correlation among sensors' measurements, and hence data compressibility.

2) The amount of data required to fit the SSAE's model depends on the underlying sensed phenomenon, and for more complex correlation patterns among sensors, more data samples are needed.



(a)



(b)

Fig. 6: Data sphering and its effects on data by showing histograms and basic statistical values. (a) Raw data in the range of $[-16.15, 47.91]$. (b) Sphered data that is scaled to a new range of $(-1, 1)$ with a Gaussian-like distribution.

B. Data Sphering

Before using historical sensor data to train the SSAE, a pre-processing operation is required, namely the data sphering stage. Data sphering is simply achieved by applying the following operation on each sensors' raw input vector $\mathbf{x} \in \mathbb{R}^N$

$$\mathbf{d} = \text{sphere}(\mathbf{x}, \sigma) = \frac{\max\left(\min\left((\mathbf{x} - \overline{x}), 3\sigma\right), -3\sigma\right)}{3\sigma} \quad (12)$$

where σ is the standard deviation of the historical training matrix $\mathbf{X}$, $\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ is the arithmetic mean of each input vector, and again $\mathbf{d} \in \mathbb{R}^N$ is the SSAE's input vector which is the resulting data vector after sphering. Unlike the standard element-wise standardization, this subtracts the arithmetic mean of each input vector and not the whole training matrix's mean value. The effect of data sphering on training data is shown in Figure 6. Clearly, the data is transformed into a smoother Gaussian-like curve with zero mean (other statistical parameters are also shown). Equally important, the resulting scale of sphered data is in the $(-1, 1)$ interval, which makes it suitable for the operation of the hyperbolic tangent function. In particular, the hyperbolic tangent function generates an output in the range of $(-1, 1)$ and without data pre-processing to this range, the SSAE cannot produce outputs similar to input data.

The reverse operation of data sphering is required at BS to reconstruct the original raw sensors' vector $\hat{\mathbf{x}} \in \mathbb{R}^N$ from the SSAE's output values $\hat{\mathbf{d}} \in \mathbb{R}^N$. The reverse operation is given as

$$\hat{\mathbf{x}} = \text{desphere}(\hat{\mathbf{d}}, \overline{x}, \sigma) = 3\sigma\hat{\mathbf{d}} + \overline{x}. \quad (13)$$

Here, σ is constant for all recovered vectors, and therefore can be stored at the BS. However, $\overline{x}$ must be sent from the GW to the BS along with the compressed data. Therefore, the transmitted data size when using CS is $M + 1$.

V. Numerical Results

In this section, we evaluate the performance of the SSAE-based sparsity inducing method.

A. SSAE Tunning

One of the main difficulties of applying neural network-based methods is the numerical tuning of the network hyper-parameters. Hyper-parameter setting of autoencoder's variants

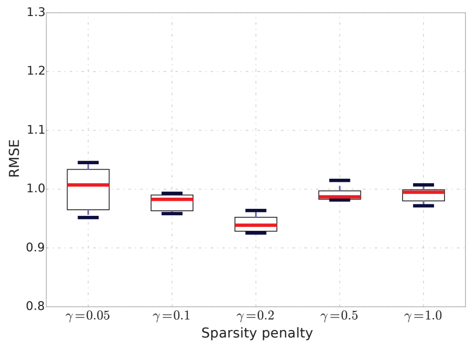


Fig. 7: Sparsity parameter setting for $\eta = 0.217$. Bars represent root mean square error (RMSE) values over 10 runs. The maximum performance achieved at $\gamma = 0.2$.

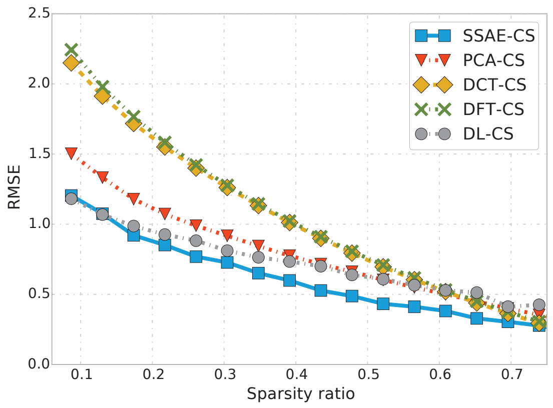


Fig. 8: Root mean square error (RMSE) versus sparsity ratio $\eta = \left(\frac{K}{L}\right)$.

can be facilitated by searching over a scale of values in the log-domain (e.g., values such as $10^{-1}, 10^{-2}, 10^{-3}, \ldots$), and then the value that minimizes the cross validation error is selected accordingly [28]. Figure 7 shows the setting of the sparsity hyper-parameter γ for sparsity ratio $\eta = 0.39$. The sparsity term in (7) can be interpreted as sparsity nomination term, that is fed to the shrinking mechanism to generate sparse codes. Therefore, trying different values of γ is useful to achieve maximum signal reconstruction performance. For SSAE in the next experiments, the following function is used for the sparsity penalty γ settings

$$\gamma(\eta) = 0.26 - 0.26\eta, \tag{14}$$

which is found by manually fitting the hyper-parameter γ for two values of η, as described above, and then finding the line connecting these two manually fitted points.

B. Comparing to Benchmarks

Using a difference matrix that captures the difference between adjacent and correlated values as a sparse basis was used in [4], [15]. Similar to [4], we noted the difference matrix's poor performance in sparsifying the data, and hence it is not included in our comparison analysis.

Figure 8 shows a comparison between the SSAE recovery performance and other conventional methods including principal component analysis (PCA), discrete Fourier transform (DFT), discrete cosine transform (DCT), and dictionary learning (DL). These conventional methods are chosen for comparison as they are widely used in the CS literature [4], [9]–[12]. Two important observations can be made.

1) Most sparsity inducing algorithms will achieve a relatively similar recovery error at high values of η. However, these high sparsity ratio values (e.g., $\eta > 0.7$) are not typical in practical applications as the reduction in data size is not noticeable. Therefore, these values cannot be used for CS's applications as the measurement vector size will be similar to the source signal size (i.e., $N \approx M$). On the other hand, SSAE significantly outperforms conventional methods for practical low sparsity ratios and when the nonzero values in the generated sparse codes are required to be minimized.
2) Conventional DL methods (e.g., [29], [30]) use the ℓ_1 minimization to model the raw data as linear combinations of sparse bases. In this paper, we used the scikit-learn library [31] for testing the dictionary learning method in which the coordinate descent method is used to find the LASSO problem solution. Similar to our algorithm, the scikit-learn's implementation enables setting the required sparsity ratio by defining the number of nonzero coefficients in the sparse code, while we set the remaining parameters to their default values. We normalize the data to a zero mean and a unit variance before learning the dictionary model. In addition to the slightly better performance, we also noticed that the learning time of the SSAE method is also shorter than the DL method. This is significant for large-scale WSNs.

C. Noisy Data

Sensors may report imprecise measurements due to external noise sources, inaccurate sensor calibration, unstable power supply, and imperfect node design [32]. In this section, we assume that noise values are independent Gaussian variables with zero mean and variance σ_z^2 such that $z \sim \mathbb{N}\left(0, \sigma_z^2 I_N\right)$, where $\mathbf{z} \in \mathbb{R}^N$ is an added noise vector. We noticed that the SSAE method does not only allows the compression of the sensors' data, but it also helps in estimating the noiseless data vector of the physical phenomenon $\mathbf{x}^* \in \mathbb{R}^N$.

An overcomplete sparse representation is achieved when the number of hidden layer's neurons (sparse code's size) is greater than the input layer's neurons (i.e., $L > N$). However, the measurement vector's size M of CS is proportional to the sparse code's size as in (3). Therefore, the number of nonzero items must be minimized, and less nonzero coefficients are defined in the overcomplete sparse code. On the other hand, using more neurons in the hidden layers can result in the overfitting problem [33]. Overfitting degrades the neural network's reconstruction performance and increases the learning time of the parameters $\boldsymbol{\theta} \doteq \left[\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\right]$. Table I summarizes the experiments of using overcomplete sparse representation. The results also include the case of adding external noise $z \sim \mathbb{N}(0, I_N)$ to sensors' measurements. This shows that

TABLE I: System performance with different numbers of hidden neurons.

L	K	γ	M	RMSE (no external noise)	RMSE (noise $\sigma_z^2 = 1$)
23	5	0.2	12	0.987	1.522 (unreliable)
25	5	0.25	12	**0.930 (best)**	1.512 (unreliable)
30	4	0.5	12	0.982 (overfitting)	**1.259 (best)**
32	4	0.6	12	1.027 (overfitting)	1.338

the overcomplete case is useful in unreliable network to reduce the noise effects while producing sparse codes. However, in noise-free networks, using overcomplete codes can degrade the sparsity-inducing algorithm performance due to the overfitting problem.

VI. Summary

In this paper, we have introduced a sparsity-inducing algorithm for data aggregation of non-sparse signal in wireless sensor networks. The proposed method consists of three steps: data collection, offline training and modeling, and online sparse code generation. The modeling scheme is based on a neural network with three layers, where the sparse codes are exposed at the hidden layer's neurons. A cost function is introduced as a sparsity nomination scheme. Then, a shrinking mechanism is used to switch off the least dominant neurons in the hidden layer, while asserting on the number of generated nonzero values in the sparse code. The resulting scheme can be used in many applications such as in compressive sensing-based data aggregation schemes.

For future research, we will analytically study the energy consumption and computational burdens of the proposed scheme.

References

[1] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu, "Path reconstruction in dynamic wireless sensor networks using compressive sensing," in *Proc. 15th ACM Int. Symp. on Mobile Ad Hoc Networking and Computing*. ACM, 2014, pp. 297–306.

[2] S.-Y. Fu, X.-K. Kuai, R. Zheng, G.-S. Yang, and Z.-G. Hou, "Compressive sensing approach based mapping and localization for mobile robot in an indoor wireless sensor network," in *Proc. Int. Conf. Networking, Sensing and Control*. IEEE, 2010, pp. 122–127.

[3] J. Meng, H. Li, and Z. Han, "Sparse event detection in wireless sensor networks using compressive sensing," in *Proc. 43rd Annu. Conf. Inform. Sciences and Syst.* IEEE, 2009, pp. 181–185.

[4] G. Quer, R. Masiero, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi, "On the interplay between routing and signal representation for compressive sensing in wireless sensor networks," in *Proc. Inform. Theory and Applicat. Workshop*. IEEE, 2009, pp. 206–215.

[5] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Proc. 21st Conf. Advances in Neural Inform. Process. Syst.*, 2008, pp. 873–880.

[6] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in *Proc. 14th Int. Conf. Artificial Intell. and Stat.*, vol. 15, 2011, pp. 315–323.

[7] Y. LeCun, "Learning invariant feature hierarchies," in *Proc. 12th European Conf. Comput. Vision*. Springer, 2012, pp. 496–505.

[8] P. Lennie, "The cost of cortical computation," *Current Biology*, vol. 13, no. 6, pp. 493–497, 2003.

[9] F. Fazel, M. Fazel, and M. Stojanovic, "Random access compressed sensing over fading and noisy communication channels," *IEEE Trans. Wireless Commun.*, vol. 12, no. 5, pp. 2114–2125, 2013.

[10] R. Masiero, G. Quer, D. Munaretto, M. Rossi, J. Widmer, and M. Zorzi, "Data acquisition through joint compressive sensing and principal component analysis," in *Proc. Global Telecommun. Conf.* IEEE, 2009, pp. 1–6.

[11] W. Bajwa, J. Haupt, A. Sayeed, and R. Nowak, "Compressive wireless sensing," in *Proc. 5th Int. Conf. Inform. Process. in Sensor Networks*. ACM, 2006, pp. 134–142.

[12] C. Luo, F. Wu, J. Sun, and C. W. Chen, "Compressive data gathering for large-scale wireless sensor networks," in *Proc. 15th Annu. Int. Conf. Mobile Computing and Networking*. ACM, 2009, pp. 145–156.

[13] A. Griffin and P. Tsakalides, "Compressed sensing of audio signals using multiple sensors," in *Proc. 16th European Signal Process. Conf.*, 2008, pp. 1–5.

[14] P. Misra, W. Hu, M. Yang, and S. Jha, "Efficient cross-correlation via sparse representation in sensor networks," in *Proc. 11th Int. Conf. on Inf. Processing in Sensor Networks*. ACM, 2012, pp. 13–24.

[15] X. Wu and M. Liu, "In-situ soil moisture sensing: Measurement scheduling and estimation using compressive sensing," in *Proc. 11th Int. Conf. Inform. Process. in Sensor Networks*. ACM, 2012, pp. 1–12.

[16] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Commun. of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[17] S. Qaisar, R. M. Bilal, W. Iqbal, M. Naureen, and S. Lee, "Compressive sensing: From theory to applications, a survey," *J. of Commun. Networks*, vol. 15, no. 5, pp. 443–456, 2013.

[18] E. J. Candes, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9, pp. 589–592, 2008.

[19] Z. Chen and J. J. Dongarra, "Condition numbers of gaussian random matrices," *SIAM J. on Matrix Analysis and Applicat.*, vol. 27, no. 3, pp. 603–620, 2005.

[20] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[21] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Commun. on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[22] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. of the Royal Statistical Soc.*, pp. 267–288, 1996.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, 1988.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[25] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. Int. Joint Conf. Artificial Intell.*, vol. 14, no. 2, 1995, pp. 1137–1145.

[26] "Sensorscope: Sensor networks for environmental monitoring," 2007. [Online]. Available: http://lcav.epfl.ch/op/edit/sensorscope-en

[27] P. Domingos, "A few useful things to know about machine learning," *Commun. of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[28] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[29] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Proc. 19th Advances in Neural Inform. Process. Syst.*, 2006, pp. 801–808.

[30] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *Proc. 26th Annu. Int. Conf. Machine Learning*. ACM, 2009, pp. 689–696.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *The J. of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava, "Sensor network data fault types," *ACM Trans. Sensor Networks*, vol. 5, no. 3, p. 25, 2009.

[33] Y. Liu, J. A. Starzyk, and Z. Zhu, "Optimized approximation algorithm in neural networks without overfitting," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 983–995, 2008.