Distributed Anomaly Detection using Autoencoder Neural Networks in WSN for IoT

Tie Luo* and Sai G. Nagarajan†
*Institute for Infocomm Research, A*STAR, Singapore
†Engineering Systems and Design Pillar, Singapore University of Technology and Design
E-mail: luot@i2r.a-star.edu.sg, sai_nagarajan@mymail.sutd.edu.sg

***Abstract*—Wireless sensor networks (WSN) are fundamental to the Internet of Things (IoT) by bridging the gap between the physical and the cyber worlds. Anomaly detection is a critical task in this context as it is responsible for identifying various events of interests such as equipment faults and undiscovered phenomena. However, this task is challenging because of the elusive nature of anomalies and the volatility of the ambient environments. In a resource-scarce setting like WSN, this challenge is further elevated and weakens the suitability of many existing solutions. In this paper, for the first time, we introduce autoencoder neural networks into WSN to solve the anomaly detection problem. We design a two-part algorithm that resides on sensors and the IoT cloud respectively, such that (i) anomalies can be detected at sensors in a fully distributed manner without the need for communicating with any other sensors or the cloud, and (ii) the relatively more computation-intensive learning task can be handled by the cloud with a much lower (and configurable) frequency. In addition to the minimal communication overhead, the computational load on sensors is also very low (of polynomial complexity) and readily affordable by most COTS sensors. Using a real WSN indoor testbed and sensor data collected over 4 consecutive months, we demonstrate via experiments that our proposed autoencoder-based anomaly detection mechanism achieves high detection accuracy and low false alarm rate. It is also able to adapt to unforeseeable and new changes in a non-stationary environment, thanks to the unsupervised learning feature of our chosen autoencoder neural networks.**

I. Introduction

Wireless sensor networks (WSN) are the eyes and ears of the Internet of Things (IoT) in the sense that they transform physical phenomena into digital signals and transfer these signals to the interconnected cyber-world for much richer processing and analytics. In reality, however, such signals or data are rarely perfect, where anomalies often arise and can interfere with the subsequent IoT analytics. *Anomalies*, also known as *outliers*, are data that do not conform to the patterns exhibited by the majority of the data set [1]. It is important to identify anomalies because they often indicate events of interest such as equipment faults, sudden environmental changes, security attacks, and so on. In fact, anomaly detection is even *essential* to many IoT systems when, oftentimes, the whole purpose of deploying a sensor network as part of an IoT ecosystem is not to know the "norm" but to capture the highly erratic event occurrences.

Although the task of anomaly detection could be undertaken by a central IoT entity such as "back-end" as often referred to, such a scheme tends to cause highly inefficient resource utilization (besides undesirable delay). This is because of the large amount of raw data that needs to be transmitted from sensors to the central entity, which entails substantial channel interference and energy consumption, while in fact only a very small fraction of the transmitted data are anomalous.

Therefore, it is more desirable to push such tasks to the "edge" as much as possible, to maximize the efficiency of resource utilization as well as responsiveness. However, anomaly detection is challenging for resource-limited sensors, owing to the elusive nature of anomalies and the volatility of ambient physical environment. Existing solutions based on various approaches have been proposed in the literature, such as threshold-based detection and Bayesian assumptions of prior distributions [2], classification using k-nearest neighbors [3], local messaging based distributed detection [4], [5], support vector machines (SVM) based detection [6], and so forth. However, they are either computationally expensive or incur large communication overhead, which weekends their applicability to resource-constrained wireless sensor networks.

In this paper, we propose to use autoencoder neural networks [7] for anomaly detection in WSN. *Autoencoders* are a deep learning model, and are traditionally used in image recognition [7] and other data mining problems such as spacecraft telemetry data analysis [8]. However, deep learning is generally not an option for WSN because of its formidable hunger for computational resources. We overcome this infeasibility by building an autoencoder neural network that consists of only three layers including the one hidden layer of neurons. This simple structure, however, performs very well due to the inherent power of reconstructing the input data by autoencoders. Specifically, we design a two-part algorithm that resides on sensors and the IoT cloud, respectively, such that (i) anomalies can be detected at sensors in a fully distributed manner, without the need for communicating with any other sensors or the cloud, and (ii) the relatively more computation-intensive learning task (for model training) can be handled by the cloud. The communication between sensors and the cloud happens at a much lower (and configurable) frequency than sensing. In addition, the computational load is also very low on sensors (in polynomial rather than exponential complexity such as in [3]), which is friendly to resource-limited sensor hardware.

For evaluation purposes, we use a sensor dataset collected over four consecutive months from a WSN testbed deployed in our office building. We demonstrate that our proposed distributed anomaly detection using autoencoders achieves high detection accuracy and low rate of false alarms (jointly characterized by a comprehensive metric called AUC). We also demonstrate that it can adapt to unforeseeable and new changes (which are however common in reality), which substantiates its suitability for non-stationary and evolving environments [9].

To the best of our knowledge, this work is the first to introduce autoencoders into WSN, or more broadly IoT, for anomaly detection. Besides the minimal communication and computation requirements and the distributed advantage, we also reap the benefit from the *unsupervised learning* feature that autoencoders possess (while most neural networks do not). This is particularly useful because it dissolves the common challenge that supervised machine learning models often lack of sufficient training data of anomalies.

II. Related Work

There are various anomaly detection methods with different levels of complexity. The basic idea is to model the distribution of what is considered to be "normal" and then check if the target data deviate from the distribution to a significant degree. For example, several statistical techniques as noted by [2] assume a prior distribution for the "normal" data and perform a hypothesis test based on that assumption. However, model mismatch will occur when data in the real world do not adhere to the assumption. Some other techniques take a threshold-based approach as also surveyed by [2], but it is difficult to identify a good threshold, and even if found, it inevitably pertains to particular setting and is hard to generalize.

In view of such problems, non-parametric methods were proposed as more sophisticated solutions. For example, [3] uses the k-nearest neighbors algorithm to create a hyper-grid around a given data point, and consider the data point anomalous if less than k other data points lie inside that hyper-grid. This algorithm has a computational complexity of $O\left(2^{M-1}\right)$, where M is the dimension of input data which can be hundreds or even thousands, making it prohibitive for high-dimensional cases.

In the quest of searching for more efficient solutions, [4] and [5] propose distributed algorithms that rely on message exchange among sensors to detect anomalies in real time. Although these methods are computationally less intensive, they require frequent and reliable communications among sensors, which imposes serious power consumption on energy-constrained sensors.

To reduce communication overhead, [6] propose two distributed online anomaly detection techniques based on a hyper-ellipsoidal one-class support vector machine (SVM). The main idea is to take advantage of the spatiotemporal correlation between sensor data and to update the SVM model (more specifically the ellipsoidal boundary) to reflect the change in the normal sensor data. However, these techniques involve matrix inversions which are computationally unfriendly to sensors. For a more comprehensive survey, the reader is referred to [9].

We propose to use autoencoder neural networks for undertaking the anomaly detection task in WSN. We leverage the reconstruction ability of autoencoders and design a two-part algorithm that eliminates the need for communication between sensors. The algorithm running on sensors only involves a simple matrix dot product operation whose computational complexity is much lower than existing solutions.

Autoencoders were traditionally used in image recognition problems [7] and spacecrafts' telemetry data analysis [8], which all have (statically) well-defined training and test data sets. In a distributed and networked context like WSN, which is much more complex and dynamic, this paper—to the best of our knowledge—is the first work that uses autoencoders for anomaly detection in a networked context.

III. Model and Method

A. Preliminaries and Model

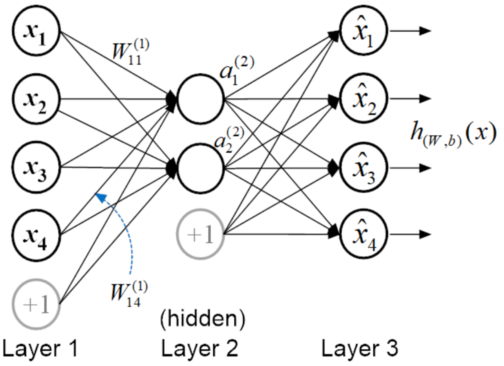


Figure 1: An autoencoder neural network.

An artificial neural network is an interconnected group of processing nodes, i.e., "neurons", that jointly perform a (typically nonlinear) transformation of inputs to certain desired outputs. An autoencoder is a special type of neural networks whose objective is to *reconstruct* the inputs instead of *predicting* some target variables. By reconstructing inputs, an autoencoder tries to learn a condensed representation of the input data, a process also known as "encoding". Formally, see Fig. 1 in which an autoencoder consists of:

1) An input layer: an M-dimension vector that represents the input signals, denoted by $\mathbf{x} = (x_1, x_2, ..., x_M)$. For example, it could be the pixel values of an image, or a time series of temperature sensor readings.
2) An output layer: denoted by vector $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_M)$. Note that this is distinct from general neural networks where we would use $\mathbf{y} = (y_1, y_2, ..., y_{|\mathbf{y}|})$ to denote the output layer. In the case of autoencoders, the output has the same dimension as the input, and we would like the output to be *equal* to the input for the purpose of *reconstructing* the original input. Hence we automatically obtain our training samples by setting $\mathbf{y} = \mathbf{x}$, which is why autoencoders are unsupervised learning models.
3) One or multiple hidden layers: sitting between the input and output layers, these hidden layers aim to learn a pattern in the inputs so as to "encode" the essential information (with affordable information loss). Let us denote the total number of all the layers by L and index the layers by $l = 1, 2, ..., L$, and denote the number of nodes in the l-th layer by $n^{(l)}$ (which does not count the *bias unit* "+1" which we explain shortly). For example in Fig. 2, $L = 3$ and $n^{(2)} = 2$.
   In general, an autoencoder has multiple hidden layers, but in this particular work we use the simplest form—one layer only—to minimize the computational overhead for distributed detection. It was generally observed to perform reasonably well in practice, as also demonstrated by our own evaluation.
4) Activation function and hyper-parameters: each neuron in layer $l = 2, 3, ..$ represents an *activation function*

$f(\cdot)$ which is typically a sigmoid function:[1]

$$f(z) = \frac{1}{1 - e^{-z}}.$$

The (hyper-)parameters of an autoencoder are weights $\mathbf{W}$ and biases $\mathbf{b}$, where $W_{ij}^{(l)}$ is the weight associated with the connection from node j in layer l to layer i in layer $l+1$,[2] and $b_i^{(l)}$ is the bias associated with node i in layer $l+1$. For example, if we denote by $a_i^{(l)}$ the output value (a.k.a. *activation*) of the neuron i in layer $l = 2, 3, ...,$ then

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + ... + W_{14}^{(1)}x_4 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + ... + W_{24}^{(1)}x_4 + b_2^{(1)})$$

In a more compact form, and more generally for all $l = 2, 3, ...,$ we write

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l-1)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l-1)}). \tag{1}$$

The intercept term $\mathbf{b}^{(l-1)}$ in the above corresponds to the "+1" node (in $n^{(l)}$ copies) at layer $l-1$, which is called a *bias unit* or bias node.

The final output, $\hat{\mathbf{x}}$, is then $\mathbf{a}^{(L)}$ which we also denote by $h_{(W,b)}(\mathbf{x})$.

Given T training samples $\{\mathbf{x}[1], \mathbf{x}[2], ..., \mathbf{x}[T]\}$ where each $\mathbf{x}[i] = (x_1[i], x_2[i], ..., x_M[i])$, the objective of an autoencoder is to minimize the cost function

$$\begin{aligned} J(W,b) :=& \frac{1}{T}\sum_{i=1}^{T}\left(\frac{1}{2}||h_{W,b}(\mathbf{x}[i]) - \mathbf{x}[i]||^2\right) \\ &+ \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{j=1}^{n^{(l)}}\sum_{i=1}^{n^{(l+1)}}\left(W_{ij}^{(l)}\right)^2 \end{aligned} \tag{2}$$

The first term defines the *reconstruction error* with respect to the original inputs, and the second term is a regularization term to prevent overfitting. Minimizing this function can be solved using standard gradient descent methods such as stochastic gradient descent or L-BFGS (Limited-memory BroydenFletcherGoldfarbShanno) algorithms [10], [11].

B. System Architecture and Two-Part Algorithm

Our system architecture is presented in Fig. 2. Each sensor runs one copy of the autoencoder and performs two tasks (apart from sensing): (i) provide the input and output of the autoencoder to the IoT cloud as the training data, which are uploaded via a gateway or cluster head in a much lower frequency than sensing (e.g., once a day versus once every two minutes), (ii) perform anomaly detection, which is done locally without the need for communicating with any other sensors, the gateway, or the IoT cloud. The gateway simply relays data between sensors and the cloud, and can be bypassed if each sensor is equipped with an Internet connection capability such as 3G, WiFi or Ethernet.

The cloud trains the autoencoder machine learning model using the training data $(\mathbf{x}, \hat{\mathbf{x}})$ provided by *all* the sensors. Afterward, it sends the updated model parameters $(\mathbf{W}, \mathbf{b})$ back to the sensors to update their respective autoencoders.[3] In addition, if there are anomalies reported by the sensors, the cloud may take actions such as alarming to a control center or conducting IoT analytics such as root cause analysis.

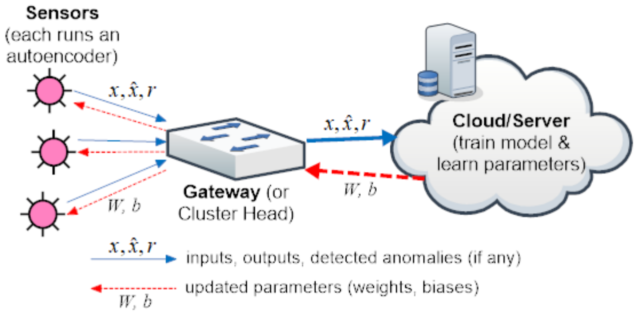


Figure 2: Architecture of a WSN that uses autoencoders for anomaly detection.

Specifically, each sensor performs anomaly detection as follows. For the ease of description, we assume the frequency of uploading training data to be once a day. Hence, if the sensing frequency is once every two minutes, then the dimension of the input vector is $M = 720$. Note that it is well known that one aggregated transmission is much more energy-efficient than multiple separate transmissions even if the data size is larger; in fact, one can reduce the communication overhead even further by using a (lossless) local data compression algorithm such as S-LZW [12] or LEC [13].

1) Each sensor, say s, calculates the reconstruction error, or *residual*, for the m-th input-output observation ($m = 1, 2, ..., M$) on a given day d:

$$r_m(s,d) = x_m(s,d) - \hat{x}_m(s,d).$$

2) Sensor s uploads $\{r_m(s,d) | m = 1, 2, ..., M\}$ to the cloud at the end of the day d.
3) The cloud calculates the statistics of the residual in terms of mean and variance:

$$\begin{aligned} \mu_m &= \frac{1}{DS}\sum_{d=1}^{D}\sum_{s=1}^{S} r_m(s,d) \\ \sigma_m^2 &= \frac{1}{DS}\sum_{d=1}^{D}\sum_{s=1}^{S} (r_m(s,d) - \mu_m)^2 \end{aligned} \tag{3}$$

where D is the total number of days over which the cloud computes the statistics, and S is the total number of sensors.
4) The cloud sends μ_m and σ_m back to every sensor.
5) Each sensor detects anomaly by calculating

$$\alpha_m(s,d) = \begin{cases} 0, & \text{if } |r_m(s,d) - \mu_m| \le p\sigma_m \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

where p is a parameter chosen according to the specific use case. Typically, the residuals r are Gaussian and we can choose $p = 2$ (or $p = 3$) which corresponds to

[1] The other common choice for f is the hyperbolic tangent, or `tanh`, function. Recently, a rectified linear activation function was discovered as yet another choice which often works well for deep learning.

[2] The reversed order of subscript indexes might look counter-intuitive but this notational convention was adopted by the neural networks literature likely because of *back-propagation*.

[3] This describes the case when the sensors are monitoring the same physical phenomenon in proximity, where at a given point in time, sensor readings are similar across sensors if no anomaly occurs. For a large-scale WSN, the sensors can be organized into c clusters (hence the "cluster head" in Fig. 2) such that each cluster is represented by an autoencoder and the cloud will simply maintain c autoencoders.

that 5% (or 2.5%) observations on the average would be classified as anomalies.

Note that step 5 does not need to wait for step 4 which simply allows sensors to update their *current* values of μ_m and σ_m.

This is essentially a two-part algorithm that resides in part on sensors and in part on the cloud. Thus, the pseudo-code of the above is presented in Algorithm 1 (DADA-S) and Algorithm 2 (DADA-C). Prior to executing the algorithms, we initialize the parameters $\mathbf{W}$ and $\mathbf{b}$ as well as $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ by training our autoencoder model over a historical data set with no or negligible anomalies (this training phase also involves presetting the "seed" values of $\mathbf{W}$ and $\mathbf{b}$ which we set as small random numbers as suggested by [11]).

Algorithm 1: DADA-S: Distributed Anomaly Detection using Autoencoders (Sensor's algorithm)

```
1 for $d \leftarrow 1$ to $\infty$ do
2     Obtain sensor readings $\mathbf{x} = \{x_1, x_2, ..., x_M\}$;
3     Feed $\mathbf{x}$ into autoencoder to obtain output
          $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_M)$;
4     Calculate residual vector $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$;
5     Detect anomaly according to (4) and obtain
          $\boldsymbol{\alpha}(s, d)$;
6     Send $\mathbf{x}, \hat{\mathbf{x}}, \mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$ to cloud (via gateway);
7     Update $\mathbf{W}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}$ received from cloud;
8 end
```

Algorithm 2: DADA-C: Distributed Anomaly Detection using Autoencoders (Cloud's algorithm)

```
1 for $d \leftarrow 1$ to $\infty$ do
2     Receive $\mathbf{x}, \hat{\mathbf{x}}, \mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$ from all the sensors
          $s = 1, 2, ..., S$;
3     Store $\mathbf{x}, \hat{\mathbf{x}}$ in the training data set and react to
          $\boldsymbol{\alpha}(s, d)$ if needed;
4     if $d$ mod $D_u = 0$ then
          // $D_u$ denotes training frequency
              (in no. of days) chosen by cloud
5         Retrain autoencoder with the updated training
              data set; // data will be
              pre-shuffled to avoid learning
              biases toward the latest data
6         Recalculate $\boldsymbol{\mu}, \boldsymbol{\sigma}$ using $\mathbf{r}(s, d), \boldsymbol{\alpha}(s, d)$
              according to (3);
7         Send updated $\mathbf{W}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma}$ to all the sensors;
8     end
9 end
```

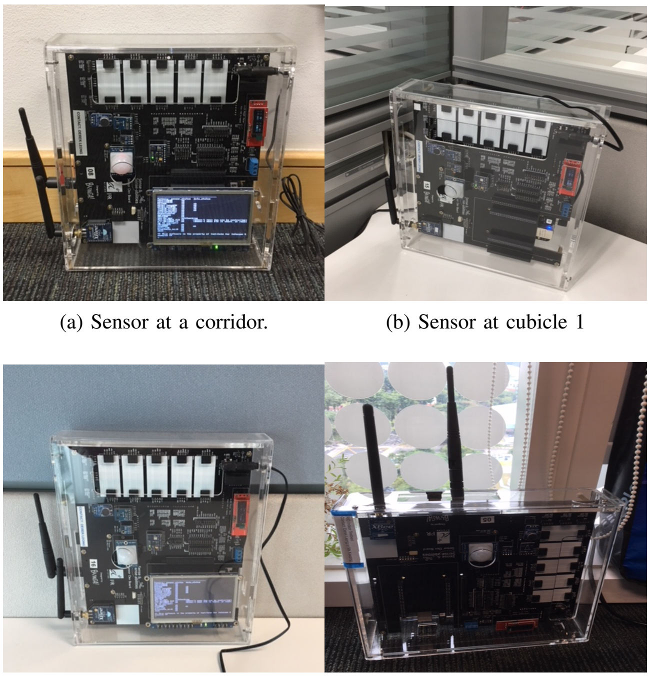
For maximal clarity, the algorithms have assumed a day to be the computational period. For mission-critical applications where anomalies need to be detected in real-time, it can be achieved by simply modifying the DADA-S algorithm such that lines 2–5 are executed when each single reading x_m is obtained.[4] Note that this does not necessarily increase communication overhead as the sensor only needs to send $\alpha_m(s, d)$ when it is equal to 1 (which only happens sporadically).

The computational complexity at sensors is dominated by Line 3 which computes the output of the autoencoder given an input vector. In fact, this only involves a simple matrix *dot product* operation (cf. (1)) with a maximum dimension of M, which has a computation time of $O(M^2)$. This is much lower than $O(2^{M-1})$ as in [3], and is readily affordable by most COTS sensors.

IV. Performance Evaluation

A. WSN Testbed and Dataset

We evaluate our proposed distributed anomaly detection algorithm using an indoor WSN testbed deployed in our office building at different locations (Fig. 3). This testbed consists of $S = 8$ sensor nodes that monitor temperature and relative humidity, with a sensing frequency of once every 2 minutes, or 720 daily readings from a single sensor. We have collected all the sensor data from September to December 2016.



(a) Sensor at a corridor.

(b) Sensor at cubicle 1

(c) Sensor at cubicle 2.

(d) Sensor beside a window.

Figure 3: The indoor deployment of our wireless sensor network.

Since it is difficult to have sufficient real anomalies, we generate synthetic anomalies into the data set using two commonly used models [14]:

- *Spike*: a sharp rise followed immediately by a sharp decline in the sensor reading. Formally,

$$x'(t) = x(t) + v\delta(t), \tag{5}$$

  where $\delta(t)$ is the Dirac delta function and v is the magnitude of the spike.
- *Burst*: a continuous and constant offset persisting for a finite period of time. Formally,

$$x'(t) = \begin{cases} x(t) + v, & t_{start} \le t \le t_{end} \\ x(t), & \text{otherwise.} \end{cases} \tag{6}$$

In both cases, v can be negative.

[4] In a highly erratic environment and for more accurate result, one may want to increase the uploading frequency so that the cloud can re-train he model using a corresponding weight sub-matrix and bias sub-array, as the order of the inputs is preserved.

B. Experimental setup

We build our autoencoder neural network with the input and output layers both having $n^{(1)} = M = 720$ nodes, and determine the number of hidden neurons, i.e., $n^{(2)}$, using k-fold cross validation. The optimal value is $n^{(2)} = 504$ which corresponds to a compression ratio of 30% $(1 - 504/720)$. We initialize the parameters $(\mathbf{W}, \mathbf{b}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ using a small portion of the dataset obtained from our WSN testbed without anomalies. In the evaluation of our algorithm, we characterize the anomaly detection performance using Area Under the Curve (AUC) corresponding to the Receiver Operating Characteristic (ROC) curves [15]. An ROC curve is most commonly used to visualize the performance of a binary classifier, in terms of true positive rate (TPR, or probability of detection) and false positive rate (FPR, or false alarm rate), and AUC is arguably the best way to summarize the classifier performance in a single number. Intuitively, AUC is the probability that a classifier assigns a higher score to a randomly chosen positive-class object than to a randomly chosen negative-class object. A good classifier has an AUC close to 1 and a bad one close to 0; an AUC of 0.5 corresponds to a random guessing classifier.

C. Results

Pre-Validation: We first verify if the autoencoder model has been trained properly, by looking at its reconstruction performance when there is no anomaly present. The results are shown in Fig. 4 for two different sensors on two different days (other sensors on other days demonstrated the similar performance). We can see that the reconstructed (or recovered) data $\hat{x}$ almost coincides with the original input x (true data), which validates our model to proceed to the next step, anomaly detection.

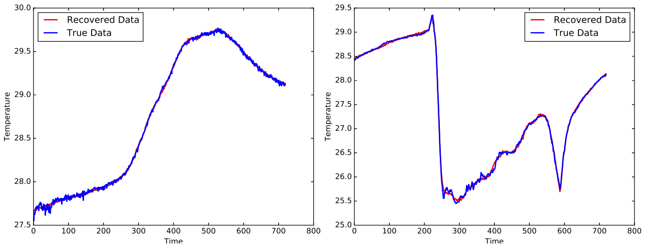


Figure 4: Reconstruction performance of the autoencoder we built. The recovered data $(\hat{x})$ almost coincides with the true data (x), indicating a valid autoencoder.

Varying anomaly magnitude: In this set of experiments, we vary the magnitude v of spikes and bursts such that v of each anomaly follows a normal distribution $\mathcal{N}(\mu_v, \sigma_v^2)$, while fixing the frequency of anomalies at $K = 100$ per day. To see how AUC is affected by two parameters μ_v and σ_v^2, we plot AUC as a heat map against both μ_v and σ_v^2 as shown in Fig. 5. We see that AUC > 0.8 most of the time, which indicates a good classifier. The exception when AUC is lower (between 0.5 and 0.8) happens when both $|\mu_v|$ and σ_v^2 are very small ($|\mu_v| < 0.07$ and $\sigma_v^2 < 0.12$). This is because in those cases, the anomalies are not really notable, or perhaps even not anomalies. Since in practice, only *significant* deviations are typically concerned with, our algorithm is suitable for all such use cases. Moreover, Fig. 5 also shows that AUC is symmetric about the mean μ_v, which is well understood since positive and negative deviations are treated the same by our neural network model (captured by residual r).

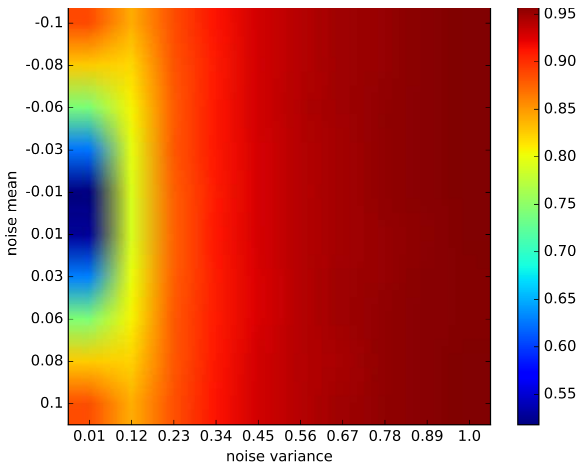
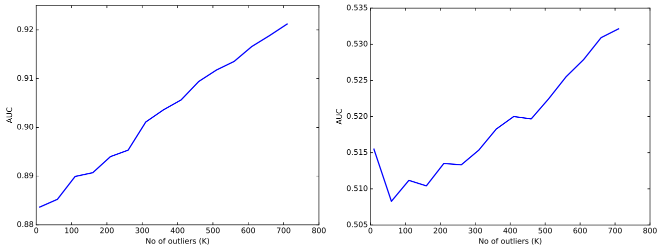


Figure 5: AUC heat map: AUC versus μ_v (Y-axis) and σ_v^2 (X-axis).

Varying anomaly frequency: In this set of experiments, we vary the frequency K, the number of anomalies per day, and present the results in Fig. 6 with the corresponding $|\mu_v|$ and σ_v^2 values. In general, when more and more anomalies occur, detecting anomalies becomes harder because the dominance of "normal" data which set the "baseline" of a classifier, is jeopardized. However, as we can see from Fig. 6, the AUC of our autoencoder continues to increase (slightly). This is because AUC is not a singular metric like misclassification error but a compound metric that summarizes both TPR and FPR. Therefore, we would rather be not too optimistic but draw a more conservative conclusion from this set of results, that the performance can be maintained within a reasonable range of K. This reflects certain *robustness* of our autoencoder.

One may wonder why the algorithm performs well even when $K = 720$, which corresponds to that every sensor reading is an anomaly. This is because there are 8 sensors in our testbed and these K anomalies are injected into the whole data set per day. In other words, even at $K = 720$, there are still 7 normal sensors on the average which help to screen out the anomalous sensor readings.



(a) $\mu_v = 0.1$, $\sigma_v^2 = 0.01$.

(b) $\mu_v = 0.01$, $\sigma_v^2 = 0.01$.

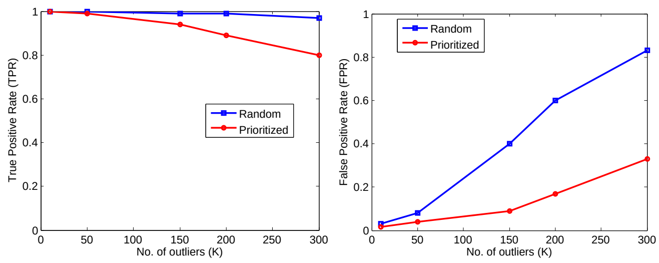
Figure 6: AUC versus the anomaly frequency K.

Adaptivity to non-stationary environment: In reality, environment is constantly evolving; observations that were previously considered anomalies could later become "normal", and vice versa. Thus, it is desirable to have an *adaptive* detector which can learn the changes and continues to work well despite unforeseeable and dynamic changes in the underlying physical phenomenon.

To this end, we configure our autoencoder with two different setups to run DADA-C:

- *Random*: Same as the above, new observations $(\mathbf{x}, \hat{\mathbf{x}})$ are randomized together with the entire historical data set which is then fed into the autoencoder to learn hyper-parameters $(\mathbf{W}, \mathbf{b})$.
- *Prioritized*: The most recent D_u days' data are mixed with anther randomly chosen D_u days of historical data to form the training data set which is then fed into the autoencoder. We set $D_u = 14$ for this evaluation.

For a clearer comparison, we evaluate true positive rate (TPR) and false positive rate (FPR) separately and show results in Fig. 7. In the evaluation, $\mu_v = 0.5$ and $\sigma_v^2 = 0.02$, and those $v > \mu_v$ are considered true anomalies while $v \leq \mu_v$ are considered acceptable environmental changes. We see that TPR slightly drops when K increases. This is because the training data set is getting more and more "perturbed" and becomes "less normal", which makes anomalies less obvious to identify. The *Random* scheme performs better (by up to 18%) because the majority of the training data is still historical data which is less affected by the new changes. On the other hand, in Fig. 7b we see that the *Prioritized* scheme has much lower false positive rate (FPR) than *Random*, by up to 60%. This is because the *Prioritized* scheme enables the autoencoder to learn from more fresh inputs so as to update weights and biases in a more responsive manner, while as the same time reserving sufficient historical data so as to keep a balance. More specifically, the variance σ_m^2 of the residual as calculated by (3) increases after each re-training, and hence allows for more room for the changes to be accepted without raising false alarms.



(a) True positive rate (TPR).

(b) False positive rate (FPR).

Figure 7: Adapting to non-stationary environment.

While the most suitable choice will depend on whether the specific application is more concerned with TPR or FPR, we could recommend *Prioritized* in general because it strikes a more balanced tradeoff. However, the key takeaway from this set of evaluation, is that our autoencoder model is able to adapt to unforeseeable and new changes in a non-stationary environment.

V. Conclusion

This paper presents the first effort of introducing autoencoder neural networks into WSN to perform anomaly detection. Despite the unsuitability of deep learning in general for resource-constrained WSN, we make this approach feasible by building a simple structure of autoencoder and exploiting its powerful reconstruct-ability. The approach is made possible also by our design of a two-part algorithm specifically for spatially distributed WSN such that communication overhead is minimized and computational load is allocated to the most suitable entities.

Specifically, our autoencoder contains a single hidden layer of neurons and the corresponding computational complexity is only polynomial ($O(M^2)$) in order to suit resource-limited sensors. Training the model involves an IoT cloud while the communication between the cloud and sensors happens at a much lower (and configurable) frequency. More importantly, anomaly detection at sensors is fully distributed and requires zero communication among sensors.

Using sensor data collected from an indoor WSN testbed over a 4-month period, we demonstrate via experiments that our proposed algorithm can detect anomalies with high accuracy and low false alarms which are jointly characterized by AUC. Furthermore, the unsupervised learning nature, as well as the flexibility in our training configuration, allows our algorithm to be able to adapt to unforeseeable and new changes in a non-stationary environment, as demonstrated in our experiments too.

In future work, we plan to extend our model to large-scale sensor networks to match the need of large-scale IoT applications seamlessly.

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[2] S. Rajasegarar, C. Leckie, and M. Palaniswami, "Anomaly detection in wireless sensor networks," *IEEE Wireless Communications*, vol. 15, no. 4, 2008.

[3] M. Xie, J. Hu, S. Han, and H.-H. Chen, "Scalable hypergrid k-NN-based online anomaly detection in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1661–1670, 2013.

[4] P.-Y. Chen, S. Yang, and J. A. McCann, "Distributed real-time anomaly detection in networked industrial sensing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3832–3842, 2015.

[5] J. W. Branch, C. Giannella, B. Szymanski, R. Wolff, and H. Kargupta, "In-network outlier detection in wireless sensor networks," *Knowledge and information systems*, vol. 34, no. 1, pp. 23–54, 2013.

[6] Y. Zhang, N. Meratnia, and P. J. Havinga, "Distributed online outlier detection in wireless sensor networks using ellipsoidal support vector machine," *Ad hoc networks*, vol. 11, no. 3, pp. 1062–1074, 2013.

[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[8] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA Workshop*. ACM, 2014, pp. 4:4–4:11.

[9] C. O'Reilly, A. Gluhak, M. A. Imran, and S. Rajasegarar, "Anomaly detection in wireless sensor networks in a non-stationary environment," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1413–1432, 2014.

[10] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA, 2009, vol. 3.

[11] Andrew Ng *et al.*, "Unsupervised feature learning and deep learning tutorial," 2012. [Online]. Available: http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

[12] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *ACM SenSys*, 2006, pp. 265–278.

[13] F. Marcelloni and M. Vecchio, "An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks," *The Computer Journal*, vol. 52, no. 8, pp. 969–987, 2009.

[14] S. Reece, S. Roberts, C. Claxton, and D. Nicholson, "Multi-sensor fault recovery in the presence of known and unknown fault types," in *12th IEEE International Conference on Information Fusion*, 2009, pp. 1695–1703.

[15] T. Fawcett, "ROC graphs: Notes and practical considerations for researchers," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 882–891, 2004.