A Machine Learning-Based Technique for False Data Injection Attacks Detection in Industrial IoT

Mariam M. N. Aboelwafa, Karim G. Seddik, *Senior Member, IEEE,* Mohamed H. Eldefrawy, Yasser Gadallah, *Senior Member, IEEE,* and Mikael Gidlund, *Senior Member, IEEE*

***Abstract*—The accelerated move towards the adoption of the industrial Internet of Things (IIoT) paradigm has resulted in numerous shortcomings as far as security is concerned. One of the IIoT affecting critical security threats is what is termed as the " False Data Injection" (FDI) attack. The FDI attacks aim to mislead the industrial platforms by falsifying their sensor measurements. FDI attacks have successfully overcome the classical threat detection approaches. In this study, we present a novel method of FDI attack detection using Autoencoders (AEs). We exploit the sensor data correlation in time and space, which in turn can help identify the falsified data. Moreover, the falsified data are cleaned using the denoising AEs. Performance evaluation proves the success of our technique in detecting FDI attacks. It also significantly outperforms a support vector machine (SVM) based approach used for the same purpose. The denoising AE data cleaning algorithm is also shown to be very effective in recovering clean data from corrupted (attacked) data.**

***Index Terms*—IIoT Security, False Data Injection Attacks, Machine Learning, Autoencoders, Support Vector Machine**

I. Introduction

THE rapid proliferation of the Internet of Things (IoT) technology in industrial enterprises has exposed the vulnerability of critical infrastructure to serious cyber-attacks. Industrial IoT (IIoT) has helped solve many intractable issues in the industry by allowing system self-controlling and offering real-time response systems. To ensure the successful roll-out of IIoT applications, security issues need to be well-studied and addressed for such applications. Specifically, IIoT sensors' readings at pivotal industrial areas are critical subjects whose loss or malfunction due to attacks or otherwise can lead to considerable losses that may include the loss of human lives [1]. For instance, the Stuxnet attack on the Iranian nuclear system [2] has led to large losses and interruptions. Monitoring of industrial systems such as hydraulic, oil, and gas stations against cyber-attacks has received great attention recently [3]. One of the major types of attacks that can affect such systems is termed the "False Data Injection" (FDI) attack. FDI attacks present a serious form of cyber-attacks against industrial infrastructures. They corrupt sensor measurements to deceive the attacked industrial platform [4]. In [5], the study demonstrates that an intruder can build an FDI attack without being detected by classical approaches such as state estimation (SE) and bad data detection (BDD) [6]. State estimation is one of the basic solutions in detecting FDI attacks in critical infrastructures. Relying on state estimation to achieve comprehensive sensing accuracy leads to measurements alteration/fabrication [7]. The study in [5] highlights this kind of weaknesses. If an intruder is aware of the measurements figures, he/she can falsify the devices readings. To address this critical vulnerability, two recent studies [8], [9], introduced machine learning (ML) using support vector machine (SVM) to efficiently detect FDI attacks. The study in [8] investigates the performance of supervised and semi-supervised machine learning approaches based on SVMs against several attacks. In their experimentation, they observed that the SVM solution performs better than the regular solutions that employ the state estimation (SE) approach for the detection of both observable and non-observable attacks. In addition, their results showed that the semi-supervised learning approaches are stronger to deal with the different data sparsity degrees than the fully-supervised learning approaches.

ML has been used in many applications to address security threats. For example, in [10], the authors present an analysis of ML techniques applied to the detection of cyber threats such as intrusion, malware, and spam. The goal is to study the current maturity of these solutions and to identify their main limitations. In [11], the authors present an ML-based technique for detecting cyber threats. The system focuses on differentiating between true positive and false positive alerts thus helping to rapidly respond to cyber threats. Another security threat, SQL injection, is studied in [12]. The authors in this survey study various machine learning algorithms used for the detection of SQL injection threats. An experimental analysis is performed in [13] of the ML methods for the Botnet DDoS attack detection. The studied methods are Support Vector Machine (SVM), Artificial Neural Network (ANN), Nave Bayes (NB), Decision Tree (DT), and Unsupervised Learning (USML). In [14], the authors cover existing security threats on ML techniques and give a survey on them from

M. Aboelwafa, K. Seddik, and Y. Gadallah are with the Electronics and Communications Engineering Department at the American University in Cairo, New Cairo, 11835 Egypt (e-mail: mariam.aboelwafa@aucegypt.edu, kseddik@aucegypt.edu, ygadallah@ieee.org). M. Eldefrawy is with the School of Information Technology, Halmstad University, 301 18 Halmstad, Sweden (e-mail: mohamed.eldefrawy@hh.se). M. Gidlund is with the Department of Information Systems and Technology, Mid Sweden University, 851 70 Sundsvall, Sweden (e-mail: mikael.gidlund@miun.se)

Parts of this work was funded by STINT under grant IB2018-7469

the aspects of the training phase and the testing phase. They also categorize the defensive techniques of machine learning into security assessment mechanisms, countermeasures, data security, and privacy.

In [9], the study presents two methods to detect false data injection. The first method uses supervised learning over labeled data and a trained SVM. The other method does not need to train data to detect the measurements deviation. In both schemes, they applied principal component analysis (PCA) to project the data on a low-dimensional space based on the observation that normal data and attacked data tend to be separated after projection. Finally, they concluded that SVM performs better than the classical attacks detection methods, such as state estimation (SE) and bad data detection (BDD) classical approaches, once it has an appropriate number of trained data sets.

In this paper, we propose to use autoencoders (AEs) as our machine-learning tool for the detection of FDI attacks. Recently, autoencoders have gained a lot of interest to address many security issues and cyber-attacks in communication networks. Autoencoders are neural networks that tend to learn a latent feature representation of the input, normally in a smaller dimension space. Another attractive feature of autoencoders is that they do not need "labeled" data for their training. In [15], the study proposes using autoencoders to learn latent features and to reduce the size of the feature vector to be fed to some machine-learning based classifier. The work in [15] considers two applications, namely, network anomaly intrusion detection and malware classification. In [16], the study proposes different anomaly detection models based on different deep neural network structures, including AEs, convolutional neural networks, and recurrent neural networks. In [17], sparse AEs are used to learn a new feature vector, of reduced size, to be fed to an SVM used for detecting network intrusion attacks. AEs are used to efficiently reduce the size of the feature vector in an unsupervised fashion. In [18], a denoising autoencoder (DAE) feeding a compact multi-layer perceptron (MLP) is used for network intrusion detection. DAEs are also used in [19] to extract some reduced size robust features of data. The same authors in [20] use stacked AEs to enhance the classification capability of neural networks. AEs are again used to learn important features to be fed to the MLP stage of the detection algorithm.

Since sensor networks are generally resource-constrained, the use of ML-based techniques for security purposes provides the additional advantage of optimizing the resource utilization for this purpose which can extend the network lifetime. Therefore, in this paper, we propose the use of an ML-based technique that uses AEs for the detection of FDI attacks. The main benefit behind the used ML-based approach is its ability to deal with data that has no structure and no model can represent it. ML-based approaches are able to extract features from raw sensor measurements. As mentioned earlier, one of the main attractive features of AEs is the fact that they can be trained in an unsupervised fashion to learn important features of the data. The AEs can learn latent correlation structures in the data samples. In our work, we exploit the correlation in two dimensions, namely, time and space (i.e. "time" correlation between the same sensor data at different times and "spatial" correlations across the sensors). AEs can learn efficient, reduced size feature representation of the data. With this feature representation, attacked data samples will show high dissimilarity between the AE input and the corresponding output.

Our approach differs from those using AEs for similar purposes (e.g. in [15]–[18]) in that we propose to use AEs as classifiers not just to learn some reduced size features. This has the benefit of simplifying the training process since there is no need for labeled data for training. Another important feature of the use of AEs as classifiers is their ability to detect other attacks since AEs are trained with unlabeled clean data. Any attack that causes a distortion in the data correlation structure would be identified by an AE-based detector. Other conventional detectors (classifiers) must be trained with labeled data sets for every attack that they have been designed to detect, and they are not guaranteed to detect any other attacks for which they were not trained. This, in addition to simplifying the training of AE-based detectors, enhances the performance of AE-based schemes in comparison to other attack detection algorithms. Moreover, it is more natural to detect false data injection attacks using AEs as they can learn latent correlation structures in the data. Dissimilarity between the expected correlation model and the observed correlation model can lead to a more natural classification compared to, for example, SVM, where it is assumed that the two classes can be separated by a hyperplane in some feature space.

The main contributions of this paper can therefore be summarized as follows:

- We propose the use of autoencoders as a classification approach to detect false data injection attacks. Autoencoders are able to learn hidden correlation structures in the data in an unsupervised manner that would allow them to detect corrupted data by assessing how far the corrupted data correlation structure is from the expected, "learned" correlation structure. Our proposed autoencoder learns correlation in two dimensions: the time (same sensor data at different times) and the spatial (across sensors) dimensions. This would allow for better representation of the correlation model at the hidden layers. To the best of our knowledge, this work is the first to propose the use of AE-based scheme to detect SDFI attacks. Additionally, our proposed scheme has the potential of detecting other types of attacks without the need for any modifications to it.
- We propose the use of denoising autoencoders to clean the corrupted data, by recovering the expected correlation structure. Our results show the efficacy of our proposed data cleaning denoising autoencoders in recovering clean data from corrupted data. To the best of our knowledge, this is the first work to propose the use of denoising AEs to clean corrupted (attacked) data.

The remainder of the paper is organized as follows. Section II presents a technical background. In Section III, we present the details of our problem description. Section IV shows the performance evaluation of our AEs machine. Section V

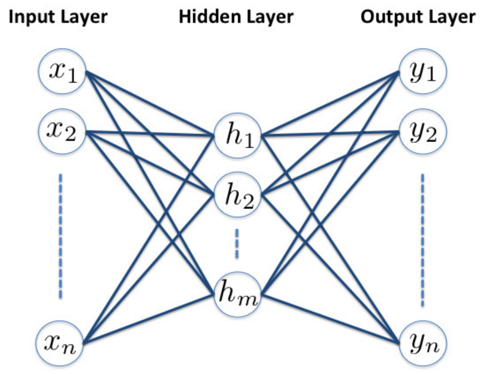

Fig. 1: Architecture of an Autoencoder with One Hidden Layer

concludes the study.

II. Technical Background

In this work, an ML-based approach is used to detect false data injection attacks. The technique relies on the use of an autoencoder (AE), which is a type of neural networks (NN) that has proven to be very effective in detecting anomalies [21]. Furthermore, a type of AEs, called denoising AE, is used to clean up falsified data by recovering the correlation structure (in the time and spatial dimensions) of sensor readings. We will compare our proposed AE-based solution to a baseline classification technique, namely, Support Vector Machine (SVM) as SVMs have been pretext of smart grids [8], [9]. In the next two subsections, a detailed technical review of the basics of AEs and SVMs is presented.

A. Autoencoders

An AE is a fully-connected, unsupervised neural network ML algorithm that applies backpropagation. As can be noticed from Fig. 1, an AE consists of an input layer, one or more hidden layers and one output layer. Generally, by letting bold letters represent vectors, the output of the neural network is represented as:

$$\mathbf{y} = h_\theta(\mathbf{x}), \tag{1}$$

where $\mathbf{x} = [x_1\ x_2\ \cdots\ x_n]^T$ is the input vector, h_θ is the full forward propagation function and θ is the set of weights and biases to be learned by the network during the training phase. This learning process is done by setting the target values at the output to be the same as the input values (i.e., $\mathbf{y} = \mathbf{x}$):

$$h_\theta(\mathbf{x}) = \mathbf{x}. \tag{2}$$

As mentioned earlier, an AE works as an unsupervised learning algorithm that applies backpropagation. *Unsupervised learning* means that during the training phase, only raw features are required with no labelling. Meanwhile, *backpropagation* means that the error between the predicted output and the target output propagates backwards from the output to each neuron in the network to update the weights based on some learning rate (α):

$$\theta_{i+1} = \theta_i + \Delta\theta_i, \tag{3}$$

where i is the index of the training epochs and $\Delta\theta_i$ is the weights update, and is calculated as

$$\Delta\theta_i = -\alpha \frac{\partial E}{\partial \theta_i}, \tag{4}$$

and E is the cost function. In our case, we use the Mean Square Error (MSE) as our error measure, which is given by:

$$E = \frac{1}{N} \sum_m (\mathbf{y}_m - \mathbf{x}_m)^2, \tag{5}$$

where N is the size of the training set, $\mathbf{x}_m$ is the m-th input vector, and $\mathbf{y}_m$ is the m-th output vector.

AEs have the ability to *learn* the structure of the data and the relationship among entries. Originally, they were used as a data-compression model. They encode a given input into a representation of smaller dimension. A decoder is then used to reconstruct the input back from the encoded compressed version. The main idea behind the compression and decompression capabilities of AEs is the fact that AEs can exploit the correlation between data entries. The more the correlated data entries are, the more compressible they become. To achieve this compression/decompression, the number of neurons in the hidden layer is constrained to be less than the number of neurons in the input/output layer[1]. The output of the hidden layer is the *compressed* (encoded) version of the input, while the final output is the *retrieved* version.

Correlation among data entries can not only be used for data compression but it can also be used for the detection of false data injection attacks. False data injection would disrupt the correlation model of the data entries. As a result, a relatively large MSE is expected at the output of the AE.

B. Denoising Autoencoder

Another type of AEs is the *Denoising Autoencoder (DAE)*. As introduced in [19], it has the same architecture of AEs. However, the principle behind DAEs is to be able to reconstruct data from an input of corrupted data. We train a DAE by corrupting the data sets and feeding them into the neural network. During the training phase, the target values at the output is set to be the original data, while inputs are the corrupted version of the data:

$$\text{target output of } h_\theta(\tilde{\mathbf{x}}) = \mathbf{x}, \tag{6}$$

where $\tilde{\mathbf{x}}$ is the corrupted set of input readings and $h_\theta(\tilde{\mathbf{x}})$ is the DAE output. DAE minimizes the cost function $E(\mathbf{x}, h_\theta(\tilde{\mathbf{x}}))$, where $E(.,.)$ is some error measure. A DAE must undo the corruption rather than simply replicating their input at the output, and in doing so it captures only the most significant features of the training data. This training enables the DAE to recover the correlation among input readings by using the original data.

[1]It should be noted that requiring the number of neurons in the AE hidden layer to be less than the number in the input/output layer is not a general requirement in AEs; some other AEs models can have the number of neurons in the hidden layer to be larger than those in the input/output layers like Sparse AEs

C. Support Vector Machines

An SVM is a classification technique that defines decision planes to separate data points from different classes. This can be visualized in Fig. 2 which shows an illustration of a 2-D example (for simplicity of illustration). The objective of the SVM is to maximize the width of the "*street*" separating the two classes. H_0 represents the boundary (hyperplane) that divides the street into two halves; H_1 and H_2 are the two planes (parallel to H_0) that touch the nearest points from each class to the boundary. These planes are given by

$$\begin{aligned} H_0 &: \mathbf{w}^T\mathbf{x} + b = 0, \\ H_1 &: \mathbf{w}^T\mathbf{x} + b = 1, \\ H_2 &: \mathbf{w}^T\mathbf{x} + b = -1, \end{aligned} \tag{7}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is input vector and b is the bias. The distance between the planes H_0 and H_1 is given by $|\mathbf{w} \cdot \mathbf{x}|/||\mathbf{w}|| = \frac{1}{||\mathbf{w}||}$. Hence, the distance between H_1 and H_2 is $\frac{2}{||\mathbf{w}||}$.

In order to maximize the margin, we therefore need to minimize $||\mathbf{w}||$, with the condition that there are no data points between the planes H_1 and H_2. This results in a constrained optimization problem that can be formulated as

$$\begin{aligned} &\min_{\mathbf{w}} \quad \frac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to :} \\ &\quad y_i(\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 \geq 0, \end{aligned} \tag{8}$$

where y_i is the label of each data point $\mathbf{x}_i$ (y_i is +1 for one class and -1 for the other class) and i is the data point iterator. Note that the constraint can be split into:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq +1 \text{ when } y_i = +1 \;\; \text{(class A)} \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 \text{ when } y_i = -1 \;\; \text{(class B).} \end{aligned} \tag{9}$$

The above optimization problem is a quadratic convex optimization problem with linear constraints, which can be solved using any quadratic programming solvers [22, Chapter 4].

SVMs can be extended to achieve a nonlinear classification by applying a kernel to the input data, mapping the input to some high-dimension feature space.

As opposed to AEs, SVMs are supervised learning-based algorithms. This means that the label of every point must be provided. Given a set of training points, each belonging to a class, an SVM training algorithm builds a non-probabilistic model that classifies new data points to one of the two classes. Based on the optimization problem described above, an SVM model represents data entries as points in a high-dimension space and finds a hyperplane to separate the classes such that the distance between the nearest data points belonging to each class to the decision boundary is maximized.

III. Problem Description

In this section, a description of the problem at hand is presented including the adopted model, the nature of the attacks and finally the proposed detection technique.

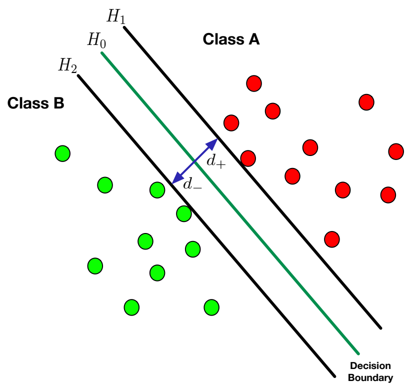

Fig. 2: Support Vector Machine Illustration

A. System Setup

The system that is used as an example in this paper is the hydraulic system presented in [3]. The data set is a group of sensor readings scattered for the hydraulic system monitoring purpose. The data set was experimentally obtained (i.e. not simulated) with a hydraulic test rig. The system cyclically repeats constant load cycles (duration of 60 seconds) and measures process values while the condition of four hydraulic components (cooler, valve, pump and accumulator) is quantitatively varied. The data set includes readings from temperature, pressure, volume flow, vibration, motor power and cooling power sensors. The total number of sensors is 15 (6 pressure sensors, 4 temperature sensors, 2 volume flow sensors, 1 motor power sensor, 1 vibration sensor and 1 cooling power sensor). The data set contains raw process sensor data (i.e. without feature extraction). The number of readings gathered from each sensor ranges from 132300 to 1323000. We use the least number of readings to avoid empty incomplete input vectors.

B. False Data Injection (FDI) Attack Description

The attack that this work aims to detect is the False Data Injection (FDI) attack. In order to adapt the use of our technique to the needs and nature of IIoT systems, we adopt the attack model presented in [23]. In this model, the intruder can alter and/or false inject single or several sensors' data at any time to have these false data within a valid range of authentic measurements.

The integrity attack on measured data is defined as follows. Let's define $\mathbf{z}_a$ as the recorded readings vector that may have some false data. Consequently, $\mathbf{z}_a$ can be described as $\mathbf{z}_a = \mathbf{z} + \mathbf{a}$, where $\mathbf{z} = [z_1, \ldots, z_m]^T$ is the clean measurements vector, and $\mathbf{a} = [a_1, \ldots, a_m]^T$ is the added/fabricated data vector. Therefore, the vector $\mathbf{a}$ represents the vector of the attack where the i-th element of $\mathbf{a}$, a_i, is of a nonzero value that reflects the intruder's ability to falsify the i-th reading and to substitute its corresponding authentic value with an alternative false value.

C. Proposed Attack Detection Algorithm

As mentioned earlier, in this study, an AE is used for attack detection due to its capability of capturing the structure of the data and learning the correlation between readings. Then, a DAE is used for cleaning the corrupted data previously detected by the AE. The process is shown in Fig. 3 which is a flowchart of the proposed scheme.

1) Autoencoder-based Detection Scheme: During the training phase, the AE is fed with the original data taken from the sensors. Target values are set to be the same data. By the end of this phase, the network becomes able to exploit the inter-correlation between entries to compress and then decompress back the input vector. Once the training phase is complete and the weights are set, the AE can then be used for false data detection.

When false data are fed to the AE, the network tries to compress and decompress the input vector. However, due to the lack of the expected correlation structure in the false data, the Mean Square Error (MSE) is larger than expected. To detect false data, the MSE is compared to a pre-determined threshold which is chosen as the mean of the validation MSE. Once an MSE value exceeds the mentioned threshold, an attack is declared. It is worth mentioning that validation is necessary to avoid overfitting which means that the machine learns the training data too well that it is unable to investigate new data. This happens when the training data keep decreasing but the validation error starts to increase. Therefore, by monitoring the validation error, early stopping of the training process occurs once it starts to increase.

2) Denoising Autoencoder-based Data Cleaning: When an attack is detected, the corrupted data are then fed into a DAE. DAEs are trained using the corrupted data as inputs and the original data as target outputs. As a result, the DAE has the capability of recovering the correlation among inputs. When falsified data are fed into the DAE, the output is a clean version of the corrupted input. This version can then be used in further processing of the system at hand as a substitute during the time of fixing the sensor under attack.

A pseudo-code of the whole process is given in Algorithm 1.

Algorithm 1 False Detection Using Autoencoder

```
1:  procedure PREPARE DATA SET
2:      Divide data set into: 60% training, 20% validation and 20% testing.
3:      Prepare a falsified copy of the testing data by replacing readings with another random numbers drawn from the normal distribution having the same variance.
4:  end procedure
5:  procedure TRAINING
6:      Initialize: Random weights.
7:      Input: $N_t$ readings from each sensor.
8:      Backpropagation Learning Technique:
9:      while Validation error is decreasing do
10:         for Each training epoch do
11:             Compute MSE based on current weights and current input: $E = \frac{1}{N}\sum_m(\mathbf{y}_m - \mathbf{x}_m)^2$, where $N$ is the size of the training set, $\mathbf{x}_m$ is the $m$-th input vector, and $\mathbf{y}_m$ is the $m$-th output vector.
12:             Update Weights based on the MSE and learning rate.
13:             Make sure that the validation error is decreasing to avoid overfitting.
14:         end for
15:     end while
16: end procedure
17: procedure TESTING
18:     Complexity: $O(d \cdot k)$ for each iteration, where $d$ is the number of dimensions of the input and $k$ denotes the number of dimensions of the encoding.
19:     Input: Testing data and falsified data.
20:     while Results are not satisfying do
21:         Change hyper parameters (e.g. learning rate, batch size, number of hidden layers, optimizer, ... etc.
22:         Repeat training phase.
23:     end while
24: end procedure
25: procedure DENOISING
26:     Training: Use corrupted data as inputs and original data as target outputs.
27:     Testing: Use detected falsified data as inputs.
28:     Output: Clean data.
29: end procedure
```

IV. Performance Evaluation

In this section, the performance evaluation results of the proposed scheme are presented. This includes comparing it to another SVM-based scheme from the literature.

A. Simulation Setup and Parameters

Simulations were carried out using the data set of [3]. The Autoencoder (AE) of the proposed scheme is trained using 60% of the data, its validation is done using 20% of the data and its testing is done using the remaining 20%. As mentioned earlier, during the training phase, target values are set equal to input values and weights are updated, one epoch after another, such that the MSE is minimized. During validation and testing, no target output is set. The output is calculated based on the weights adjusted by the training and finally the MSE is calculated. The validation error is used for two purposes. First, it is monitored to avoid overfitting. Second, it is used to calculate the threshold which, if exceeded by the MSE, an alarm (detected attack) is set. Testing error is compared (for every input) with the threshold to decide whether the input data are attacked. It is worth noting that AEs need be exposed to neither the false data during the training phase nor the labels for the input training data.

To exploit both the inter-correlation between the sensors and the autocorrelation of the readings of a single sensor in the time domain, the input to the AE is set to include more than one reading from each sensor at every iteration rather than a single reading per sensor. To choose the best number of time instants to be considered, the AE was trained and tested for $N_t = 1,\ 2,\ 3,\ 4,\ 5,$ and 10, where N_t is the number of readings (time instants) from each sensor to be fed to the AE per epoch. The training loss for each N_t value is shown in Fig. 4. It is clear that the lowest loss value is at $N_t = 2$, where it is 3.99e-7 for the training loss and 4.37e-7 for the validation loss. The trend of decreasing training and validation losses at $N_t = 2$ can be seen in Fig. 5.

The readings from each sensor are fed to the AE in parallel to other sensors. There are 15 sensors in the network and each one feeds the AE with N_t consecutive readings every iteration. This results in a total of $15 * N_t$ neurons in the input layer.

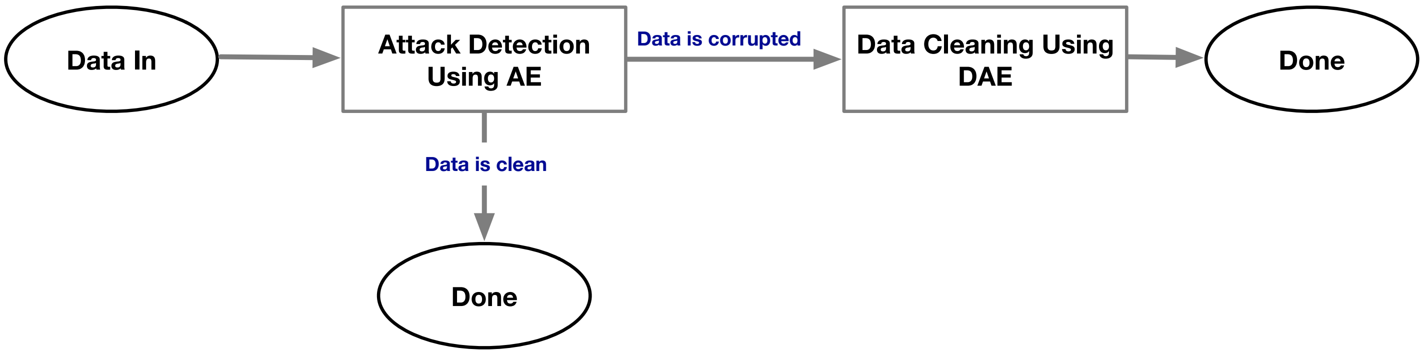

Fig. 3: Flowchart of the AE-based Proposed Solution.

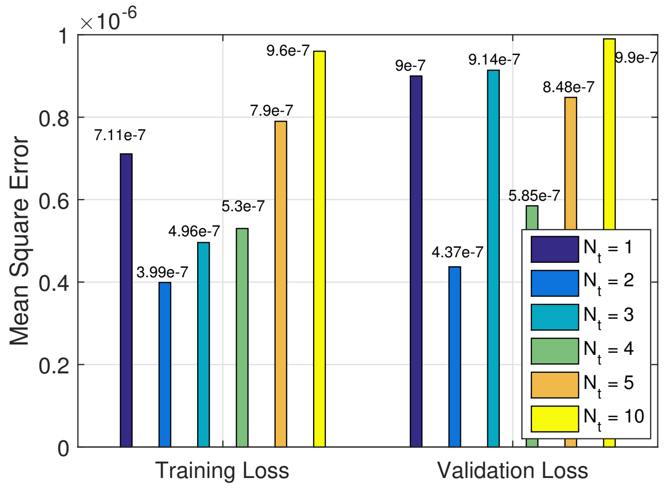

Fig. 4: The Effect of Varying Number of Readings per Sensor on the Training and Validation Losses

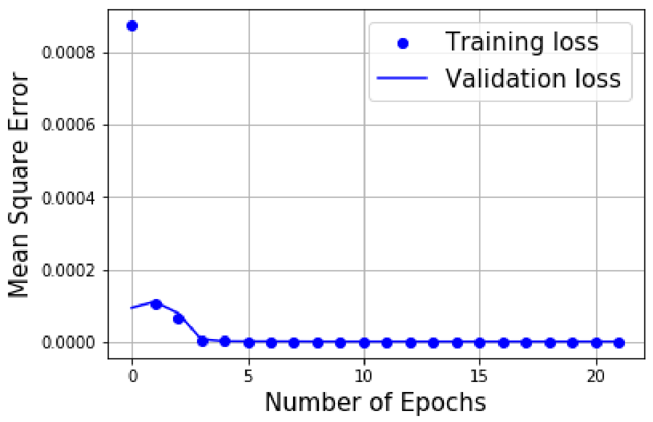

Fig. 5: Training and Validation Losses in the Training Phase.

As mentioned earlier, the output layer is the same as the input layer. Five hidden layers are used. The compression factor, which is the ratio between the number of inputs and the innermost hidden layer which stands between the encoder and the decoder, is set to 3. The size of the hidden layers between the input (output) layer and the innermost layer decreases linearly according to the compression factor. It is worth mentioning that the number of hidden layers and the compression factor are hyper parameters that are determined after several trials. A summary of the AE simulation parameters is listed in Table I.

When considering the comparison with SVM, on the other hand, we find that SVM must be trained with both types of data (original and false) along with labels for each class. This requires some extra effort for preparing the false (attacked) data for training and labeling each set of sensors' data. During the training phase, the machine creates the optimization model (based on Lagrange multipliers). Then, during the testing phase, the model investigates new inputs and classifies each of the classes (false data class or clean data class).

Falsified data used for testing and also for SVM training are obtained by two methods

1) Add a random number to the entries of all sensors. The random number is drawn from a uniform distribution in the range of $+/-10\%$ of the original value - Referred to as "Case 1" in the rest of this section.
2) Alter the entries of only one sensor (to simulate the case of only one attacked sensor). The entries of one sensor are replaced with another value drawn from a normal distribution with mean and variance that are equal to those of original data (To create a scenario more difficult to be detected). This case is referred to as "Case 2" in the rest of this section

Both machines (AE and SVM) were tested under the two scenarios.

As mentioned earlier, the AE was tested for $N_t = 1, 2, 3, 4, 5,$ and 10. For the sake of further confirmation of the best choice of N_t, an actual test under "Case 1" and "Case 2" is carried out for each value of N_t. Furthermore, the test was done for the SVM as well to make a fair comparison. Results can be seen in Fig. 6 in which the percentage of accurate decisions is plotted. In the figure, the *Accuracy Rate* can be defined as the percentage of the number of entries that were correctly decided by the machine (regardless of whether an attack or clean data were determined) to the whole number of entries. From the figure, we can confirm that $N_t = 2$ is the best choice for the proposed AE-based scheme. However, for the SVM, $N_t = 3$ seems to give better results. For this reason, further comparisons will take place using the best value of N_t for the corresponding scheme. It can also be seen from Fig.

TABLE I: Autoencoder Simulation Parameters

Parameter	Value
Number of readings per sensor (N_t)	[1 2 3 4 5 10]
Number of neurons in input layer	$N_t * 15$
Number of neurons in output layer	$N_t * 15$
Compression factor	3
Number of hidden layers	5
Number of neurons per hidden layer	Decreases linearly according to the compression factor
Learning Algorithm	Unsupervised learning applying backpropagation
Optimizer	RMSprop
Learning Rate	0.001
Training Data	60% of whole set
Validation Data	20% of whole set
Testing Data	20% of whole set + equal amount of false data
Number of training epochs	20 (determined by the early stopping criterion)
Threshold	mean of validation error

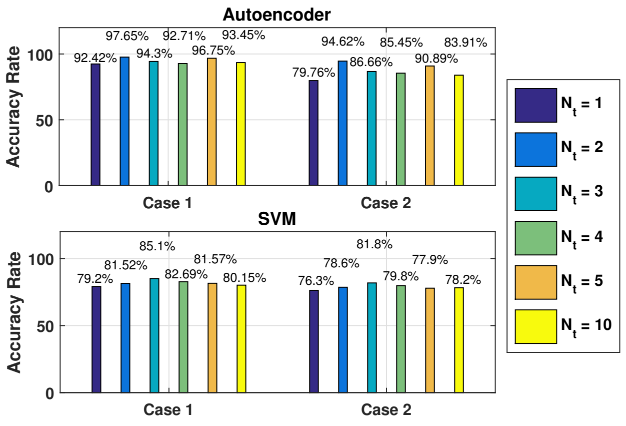

Fig. 6: The Effect of Varying Number of Readings per Sensor on the Decision Accuracy

6 that the accuracy in “Case 1” (for both machines) is higher than that of “Case 2”. This is due to the difficulty of detecting only one malfunctioning (attacked) sensor. Moreover, the type of attack is even trickier to be detected. This is because replacing the sensor readings with values from the normal distribution with the same mean and variance will slightly affect the inter-correlation between readings. It is also worth mentioning that the accuracy of the proposed scheme is higher than that of the SVM-based scheme in all cases. However, this will be discussed in more details later.

B. Testing and Comparing the Final Decision Accuracy

A comparison between the two machines from the perspective of the percentage of correct attack detection and the percentage of false alarm is shown in Fig. 7. In the figure, the *Rate of Detection* can be defined as the percentage of the number of false entries that were correctly detected by the machine to the whole number of entries. Additionally, the *Rate of False Alarm* can be defined as the percentage of the number of clean entries that were mistakenly decided by the machine as false data to the whole number of entries. Other famous kernels (other than the linear kernel) were tested for the SVM (e.g., RBF and Gaussian). The AE has proven to be more reliable and it outperforms the SVM in both cases. This can be justified by the capability of AEs to learn the hidden complex correlation structures of the data. Anomalies in input data that cause the difference between the expected data structure and the observed one can lead to a more robust classification compared to SVM where it is assumed that the two classes are separable by a hyperplane in some space.

A higher percentage of detection and a lower percentage of false alarm are recorded. Furthermore, the percentage of detection of the AE-based scheme reaches 100% in “Case 1”. SVM, with a linear kernel, gives better performance than with the other two kernels. It is worth mentioning that the AE gives equal percentages of false alarm in both cases. This is because during the training phase, the machine was subjected to original (clean) data only and the same machine is used for both cases. However, the SVM must be subjected to false data in the training phase. That is why each case needs a new trained machine. This is also another major advantage of AE-based approaches, which is that they can detect other attacks since AEs are not trained to classify any “specific” attack. Rather, AEs are trained with clean data. An AE-based attack detector can detect any attack that causes a significant distortion in the data correlation model. However, to detect other attacks in the case of SVM, the machine must be trained with labeled data for every possible attack and there is no guarantee that an SVM trained to classify a specific attack will be able to detect other attacks.

To make a better visualization of the results, Fig. 8 gives a snap shot of a portion of the time series of the test data and the decision of both machines. As explained earlier, the ability of the AE to learn the data structure results in better performance and higher accuracy. Therefore, Fig. 8 shows less miss-detections and false alarms in the case of the AE-based scheme. It is worth mentioning that the AE is not better than the SVM at every instant. For example, at instant 175, SVM was able to detect the attack while AE was not. However, AE still gives a better average performance and a higher percentage of detection.

The Receiver Operating Characteristics (ROC) plot for both machines is shown in Fig. 9. The ROC is the relation between the rate of false alarm and the rate of detection. When

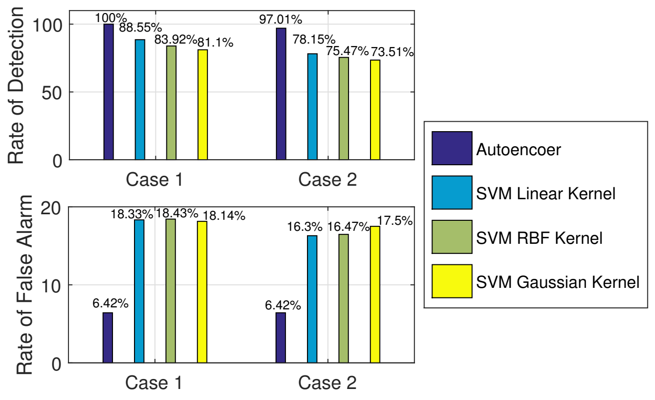

Fig. 7: A comparison w.r.t Detection Percentage and False Alarm Percentage

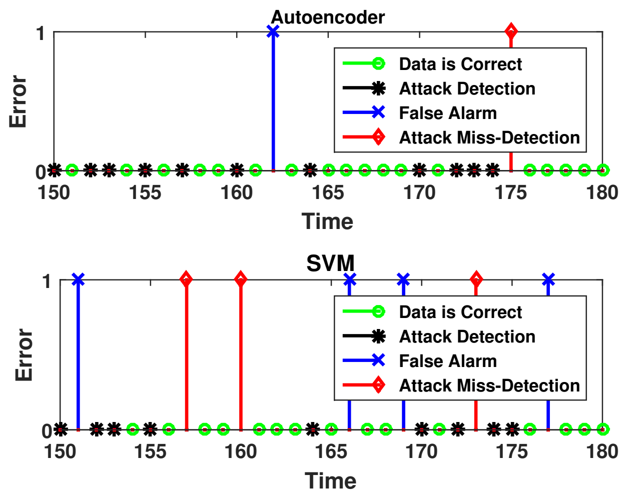

Fig. 8: A Visualization of a Time Series Snap Shot

comparing two methodologies, a better scheme is the one that achieves a higher detection rate for the same false alarm rate. It is noticeable from Fig. 9 that the AE-based scheme outperforms the SVM-based scheme in terms of ROC as well.

C. Complexity Analysis

The training complexity of SVM can be approximated to $O(N^3)$, where N is number of data entries [24]. On the other hand, the training complexity of AEs is $O(d \cdot k)$ for each iteration, where d is the number of dimensions of the input and k denotes the number of dimensions of the encoding [25]. To apply this to the case at hand, the number of readings taken from each sensor is 132300. Only 60% is used for training; this gives 79380. By taking $N_t = 3$ readings from each sensor every iteration, the total number of entries is $\frac{79380}{3}(= 26460)$ for each sensor. With 15 sensors under consideration, this results in $N = 396900$ data entries. This gives a training complexity, approximately equals $O(6e16)$ for the SVM. On the other hand, for the AE, the dimension of input vector $d = N_t * 15 = 30$, while the dimension of the encoded vector $k = 10$ (since the compression factor is chosen to be 3). It is also necessary to consider the number of training epochs, which is chosen to be 20 in our case. This results in a training complexity $O(30 * 10 * 396900 * 20) = O(2e9)$. Clearly, the complexity of the AE is a lot less than that of the SVM. The training was carried out on a SAMSUNG laptop that uses a Windows 10 Pro operating system, an Intel 2.40 GHz Core i7 processor and 8.00 GB RAM. The AE needed less than one minute to train while the SVM (linear kernel) needed 15 minutes to train.

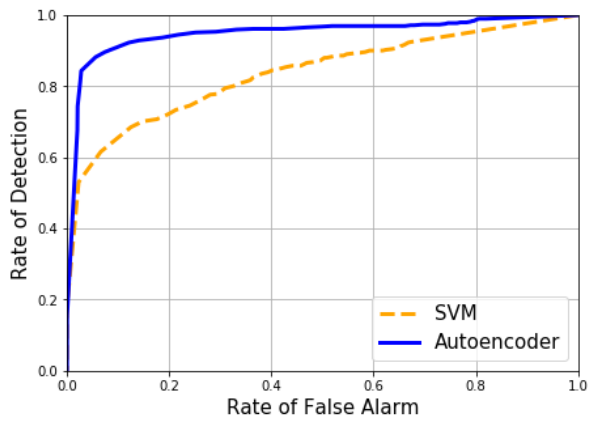

Fig. 9: Receiver Operating Characteristics Plot

D. Denoising Autoencoder

In this experiment, the Denoising Autoencoder (DAE) is used to clean up the corrupted data, detected by the original AE. The DAE is trained and tested using corrupted data from both cases (Case 1 and Case 2). As mentioned earlier, when an attack is detected (by the AE), the falsified data are fed to the DAE to obtain a clean version as close as possible to original data. To prove that the output of the DAE is able to regain the correlation model of the data, the Mean Square Error (MSE) of the corrupted data (both cases) is plotted before and after the DAE in Fig. 10. This *error* represents the difference between the corrupted data and the original data with and without the DAE. In Fig. 10, the MSE is noticed to decrease immensely when DAE is used. This means that the DAE is able to recover a clean version of the corrupted data by recovering the data "hidden" correlation model. It is also noticeable that the MSE in case 2 is less than that in case 1. This is because the nature of the attack in case 1 causes all sensors to be corrupt, while in case 2, only one sensor is corrupt. Therefore, although the attack on only one sensor is harder to detect, if the attack is causing only one sensor to be corrupted it is easier to recover the data if the attack is detected. It is worth mentioning that the input data are normalized, that is why mean square errors are all less than 1.

E. Limitations

In the following we highlight the limitations of the proposed algorithm. These limitations can be summarized as follows

- The proposed algorithm is able to detect attacks that ruin the correlation among sensor readings. If the attack is able to maintain this correlation, the AE might not be able to detect it. For instance, if the attacker used data from the measurements history in which readings are taken from the same cycle, the AE algorithm may not detect this.

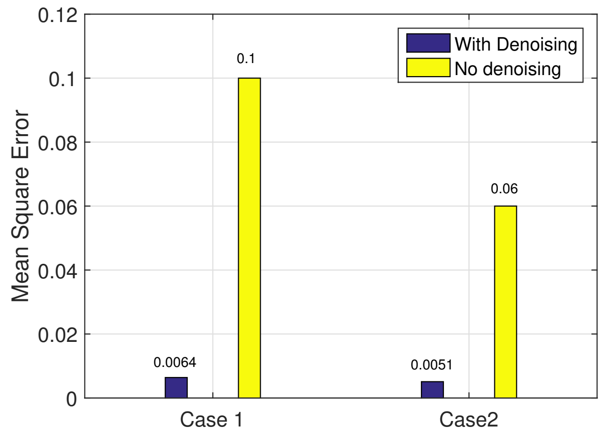

Fig. 10: Mean Square Error of the corrupted data before and after the Denoising Autoencoder.

- The DAE must be trained for each type of attack, unlike the AE which does not need any labels or supervised learning. This limits the use of the DAE to the specific attacks that it was trained to fix.

V. Conclusions

In this paper, we presented a novel method of detecting false data injection attacks against a complex hydraulic sensor-based system using Autoencoders (AEs). The proposed detection method offers better detection performance compared to support vector machine based methods (e.g. linear kernel, RBF kernel, and Gaussian kernel). In addition, AEs are easier to train since they do not require labeled data for training. Finally, AEs are able to detect different attacks since they can learn hidden complex correlation structures in the data. Any attack that will cause significant changes in these correlation structures can be detected by the AE-based attack detection algorithm. However, this is not the case for any other classifier like SVM which is trained to detect a "specific" attack (or a set of attacks) and it is not guaranteed to detect any other attack for which it was not trained with labeled data. We adopted two different scenarios of FDI attacks which we referred to as case 1 and case 2 in the study. In both scenarios, our approach offered the highest probability of attack detection with the lowest rates of false alarm and the lowest execution time. We also presented the results of using a denoising autoencoder to recover from the attack's effects on data. The results demonstrated the high ability of the denoising autoencoder to recover the data to their original state with very low mean square error values.

References

[1] S. Forsström, I. Butun, M. Eldefrawy, U. Jennehag, and M. Gidlund, "Challenges of securing the industrial internet of things value chain," in *2018 Workshop on Metrology for Industry 4.0 and IoT*. IEEE, 2018, pp. 218–223.

[2] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 4490–4494.

[3] N. Helwig, E. Pignanelli, and A. Schütze, "Condition monitoring of a complex hydraulic system using multivariate statistics," in *Instrumentation and Measurement Technology Conference (I2MTC), 2015 IEEE International*. IEEE, 2015, pp. 210–215.

[4] M. A. Rahman and H. Mohsenian-Rad, "False data injection attacks with incomplete information against smart power grids," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. Citeseer, 2012, pp. 3153–3158.

[5] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 21–32.

[6] A. Anwar, A. N. Mahmood, and M. Pickering, "Modeling and performance evaluation of stealthy false data injection attacks on smart grid in the presence of corrupted measurements," *Journal of Computer and System Sciences*, vol. 83, no. 1, pp. 58–72, 2017.

[7] A. Ashok, M. Govindarasu, and V. Ajjarapu, "Online detection of stealthy false data injection attacks in power system state estimation," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 1636–1646, 2018.

[8] M. Ozay, I. Esnaola, F. T. Y. Vural, S. R. Kulkarni, and H. V. Poor, "Machine learning methods for attack detection in the smart grid," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 8, pp. 1773–1786, 2016.

[9] M. Esmalifalak, L. Liu, N. Nguyen, R. Zheng, and Z. Han, "Detecting stealthy false data injection using machine learning in smart grid," *IEEE Systems Journal*, vol. 11, no. 3, pp. 1644–1652, 2017.

[10] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *2018 10th International Conference on Cyber Conflict (CyCon)*. IEEE, 2018, pp. 371–390.

[11] J. Lee, J. Kim, I. Kim, and K. Han, "Cyber threat detection based on artificial neural networks using event profiles," *IEEE Access*, vol. 7, pp. 165 607–165 626, 2019.

[12] T. Pattewar, H. Patil, H. Patil, N. Patil, M. Taneja, and T. Wadile, "Detection of sql injection using machine learning: A survey," 2019.

[13] T. A. Tuan, H. V. Long, R. Kumar, I. Priyadarshini, N. T. K. Son *et al.*, "Performance evaluation of botnet ddos attack detection using machine learning," *Evolutionary Intelligence*, 2019.

[14] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," *IEEE access*, vol. 6, pp. 12 103–12 117, 2018.

[15] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 3854–3861.

[16] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48 231–48 246, 2018.

[17] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with svm for network intrusion detection," *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[18] H. Zhang, C. Q. Wu, S. Gao, Z. Wang, Y. Xu, and Y. Liu, "An effective deep learning based scheme for network intrusion detection," in *2018 24th International Conference on Pattern Recognition (ICPR)*, Aug 2018, pp. 682–687.

[19] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[20] 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.

[21] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 665–674. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098052

[22] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[23] Y. Wang, Z. Xu, J. Zhang, L. Xu, H. Wang, and G. Gu, "Srid: State relation based intrusion detection for false data injection attacks in scada," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 401–418.

[24] A. Abdiansah and R. Wardoyo, "Time complexity analysis of support vector machines (svm) in libsvm," *International Journal Computer and Application*, 2015.

[25] O. Irsoy and E. Alpaydin, "Autoencoder trees," in *Asian Conference on Machine Learning*, 2016, pp. 378–390.