A Novel Anomaly Detection Algorithm for Hybrid Production Systems based on Deep Learning and Timed Automata

Nemanja Hranisavljevic[1] and **Oliver Niggemann**[1,2] and **Alexander Maier**[2]
[1]inIT - Institute Industrial IT, Lemgo, Germany
e-mail: {nemanja.hranisavljevic@stud.hs-owl.de, oliver.niggemann@hs-owl.de}
[2]Fraunhofer Application Center Industrial Automation IOSB-INA, Lemgo, Germany
e-mail: {oliver.niggemann, alexander.maier}@iosb-ina.fraunhofer.de

Abstract

Performing anomaly detection in hybrid systems is a challenging task since it requires analysis of timing behavior and mutual dependencies of both discrete and continuous signals. Typically, it requires modeling system behavior, which is often accomplished manually by human engineers. Using machine learning for creating a behavioral model from observations has advantages, such as lower development costs and fewer requirements for specific knowledge about the system.

The paper presents *DAD:DeepAnomalyDetection*, new approach for automatic model learning and anomaly detection in hybrid production systems. It combines deep learning and timed automata for creating behavioral model from observations. The ability of deep belief nets to extract binary features from real-valued inputs is used for transformation of continuous to discrete signals. These signals, together with the original discrete signals are than handled in an identical way. Anomaly detection is performed by the comparison of actual and predicted system behavior. The algorithm has been applied to few data sets including two from real systems and has shown promising results.

1 Introduction

The analysis and diagnosis of complex production plants has gained new attention due to research agendas such as Cyber-physical Production Systems (CPPSs) [1; 2], the US initiative Industrial Internet [3] or its German pendant "Industrie 4.0". In these agendas, a major focus is on the self-diagnosis capabilities for complex and distributed CPPSs. Typical goals of such self-diagnosis approaches are the detection of anomalies, of suboptimal energy consumption, of error causes or of wear [4; 5; 6].

Typical challenges for the diagnosis of CPPSs are the increasing complexity of plants, the high frequency of plant modifications—mainly due to product variants—and non-trivial causalities and timing dependencies. A typical CPPS solution for these challenges is a data-driven and machine learning based approach, which complements the traditional solutions based on manual models [7]. This places data analysis and machine learning algorithms at the heart of CPPS, Industrial Internet and Industrie 4.0. The underlying research question could be stated as follows: Can we extract models of plant behavior, of causalities and of timings, from measured data? Once such models have been learned, anomalies can be identified easily by comparing observations to the learned normal, error causes can be computed by back-tracing the causalities leading to the observations or suboptimal energy consumption can be detected by comparing energy consumption at different operating points.

So far, a large number of heterogenous machine learning algorithms have been used for this: e.g. state machines are learned for discrete variables such as control signals [8], time series have been learned for the timing behavior of continuous signals [9] or statistical models have been learned for dependent variables [10]. However, in all these cases, discrete and continuous signals heve been analyzed separately and differently, rendering a holistic signal analysis impossible. Even if hybrid automata are learned [8], the states will still be defined by discrete signals and continuous signals will be dealt with separately within the states.

Here, the new algorithm *DAD:DeepAnomalyDetection* is introduced which handles discrete and continuous signals in a uniform way and is able to analyze the system's overall timing. The main idea of the algorithm is shown in an abstract manner in figure 1 (details are given in Section 5): First, a sliding time window of the set of continuous data

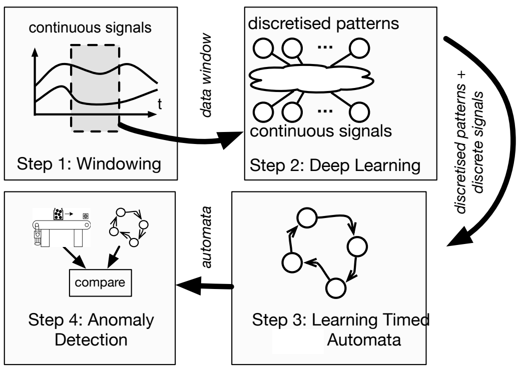

Figure 1: The main idea of *DAD:DeepAnomalyDetection*.

is used as the input layer of a deep learning network (step 1). In step 2, this network learns reoccurring patterns in the data. These reoccurring patterns result in a specific binary pattern on the top layer of the deep learning network, i.e. the network discretizes the continuous signals. Now the observations can be interpreted as a sequence of discrete patterns or events—and can be combined with the discrete signals from the observations. In step 3, from these sequences of patterns or events a timed automata is learned. At this point, we can handle the set of all signals in a uniform discrete manner, this corresponds to a human operator who understands a system behavior as a causal model of reoccurring patterns. Finally, in step 4, anomalies can be detected by comparing the predictions of the learned automata with the actual observations. Typical anomalies are unusual timings, unexpected events or new patterns.

The main contribution of this paper is threefold. First of all, for the first time (to the knowledge of the authors), a deep learning network is used for unsupervised anomaly detection of production systems. Second, based on the deep learning network, a new anomaly detection algorithm *DAD:DeepAnomalyDetection* is introduced which handles continuous and discrete signals in a uniform manner. Third, the approach is evaluated using both real systems and artificial data.

The paper is structured as follows: Section 2 gives preliminary definitions. In Sections 3 and 4 we give an overview and formulate concepts from deep learning and timed autamata theory that are referred to in Section 5, where the outline and formal definition of the new algorithm is presented. Section 6 presents some experimental results, while the paper is concluded in Section 7.

2 Preliminary definitions

In this section we give several definitions which we will refer to later in the paper.

Definition 1 (Signal vector)**.** *Signal vector is a vector* $\boldsymbol{u} = (u_1, u_2, ..., u_{|\boldsymbol{u}|})^T$*, where each* $u_i|_{i\in\{1,...,|\boldsymbol{u}|\}}$ *is a value of a discrete* ($u_i \in \{0,1\}$) *or continuous* ($u_i \in \mathbf{R}$) *signal and* $|\boldsymbol{u}| \in \mathbf{N}$ *is a number of signals in an observed system.*

Definition 2 (Observation example)**.** *An observation example is a set of pairs* $O = \{(t_1, \boldsymbol{u}_1), (t_2, \boldsymbol{u}_2), ..., (t_{|o|}, \boldsymbol{u}_{|o|})\}$*, where* $\boldsymbol{u}_i|_{i\in\{1,..,|o|\}}$ *is a signal vector according to Definition 1,* $t_i \in \mathbf{R}_{\geq 0}|_{i\in\{1,..,|o|\}}$ *is the corresponding time stamp and* $|o| \in \mathbf{N}$ *is the number of observations in O.*

Definition 3 (Snapshot)**.** *A snapshot is a column vector* $\boldsymbol{s}$*, result of windowing method applied on observation example O (Definition 2). Windowing of observation example O at moment* $t_w \in \mathbf{R}_{\geq 0}$ *using window of length* $|\boldsymbol{s}| \in \mathbf{N}$ *and sample time* $T_s \in \mathbf{R}_{\geq 0}$ *gives snapshot* $\boldsymbol{s} = (\boldsymbol{u}_1^T||...||\boldsymbol{u}_{|\boldsymbol{s}|}^T)^T$*, where* $\boldsymbol{u}_i|(t_i, \boldsymbol{u}_i) \in O, t_i = t_w + (i - |\boldsymbol{s}|)\cdot T_s, i \in \{1, ..., |\boldsymbol{s}|\}$*.* $||$ *is a symbol for vector concatenation.*

3 Deep learning

Deep learning refers to a class of machine learning algorithms used to learn deep architectures, where depth of an architecture refers to the number of levels of composition of nonlinear operations in the function learned by the model. Deep learning among other methods, includes training multilayer perceptrons, deep belief nets and deep auto-encoders.

Typically, deep architectures transform raw input representation of data into gradually more and more abstract representations. We consider that modeling time behavior and mutual dependencies of a large number of signals in a production plant can be done efficiently using deep architectures. Furthermore, we want to make use of often available, large data sets of observations from production plants and find compressed binary representation of plant condition in time. *DAD:DeepAnomalyDetection* algorithm relies on a restricted Boltzmann machine and deep belief net which will be described in the following sections.

3.1 Restricted Boltzmann machine

A restricted Boltzmann machine (RBM) is an undirected probabilistic graphical model with two layers of units (stochastic variables): visible (input) $\mathbf{v}$ and hidden $\mathbf{h}$ as illustrated on figure 2. RBM can learn a model of data distribution from training examples presented to its inputs and it is a building block of deep belief net (Section 3.2). There are different variants of RBM, while we use Bernoulli-Bernoulli (BBRBM) and Gaussian-Bernoulli restricted Boltzmann machine (GBRBM):

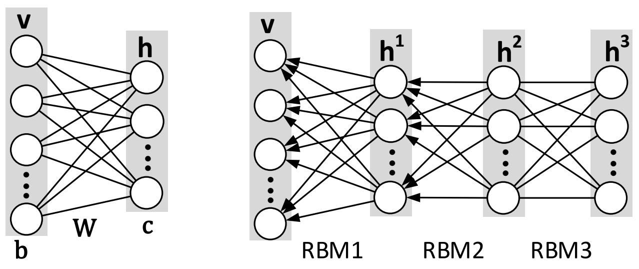

Figure 2: Undirected graphical model of RBM (left) and DBN with three hidden layers (right)—connections between units show their specific conditional dependence structure

Definition 4 (Restricted Boltzmann machine)**.** *A RBM is an undirected graphical model that models joint probability distribution of visible units* $\boldsymbol{v} = (v_1, ..., v_m)^T$ *and hidden units* $\boldsymbol{h} = (h_1, ..., h_n)^T$*. It is a tuple* $\mathcal{RBM} = (W, b, c, p(\boldsymbol{h}|\boldsymbol{v}), p(\boldsymbol{v}|\boldsymbol{h}))$*, where:*

- $W \in \mathbf{R}^{m\times n}$ *is weight matrix with each element* $W_{i,j}$ *describing dependence between* v_i *and* h_j
- $b \in \mathbf{R}^m$ *is column vector with each element* b_i *representing bias of visible unit* v_i
- $c \in \mathbf{R}^n$ *is column vector with each element* c_j *representing bias of hidden unit* h_j
- $p(\boldsymbol{h}|\boldsymbol{v})$ *is probability function of hidden given visible units.* $\{h_j|_{j\in\{1,...,n\}}\}$ *are conditionally independent given* $\boldsymbol{v}$*.* $p(h_j|\boldsymbol{v})$ *is parameterized by W, c and is member of exponential family distributions.*
- $p(\boldsymbol{v}|\boldsymbol{h})$ *is probability function of visible given hidden units.* $\{v_i|_{i\in\{1,...,m\}}\}$ *are conditionally independent given* $\boldsymbol{h}$*.* $p(v_i|\boldsymbol{h})$ *is parameterized by W, b and is member of exponential family distributions.*

Definition 5. *(Bernoulli-Bernoulli RBM) A BBRBM is a RBM for which:*

- $\boldsymbol{v}$ *takes values from* $\{0,1\}^m$
- $\boldsymbol{h}$ *takes values from* $\{0,1\}^n$
- *Conditional expectation of hidden given visible layer is* $\mu^h = E[\boldsymbol{h}|\boldsymbol{v}] = \sigma(c + W^T\boldsymbol{v})$ *and* $p(h_j|\boldsymbol{v}) = \mathcal{B}(\mu^h_j)$.[1,2]
- *Conditional expectation of visible given hidden layer is* $\mu^v = E[\boldsymbol{v}|\boldsymbol{h}] = \sigma(b + W\boldsymbol{h})$ *and* $p(v_i|\boldsymbol{h}) = \mathcal{B}(\mu^v_i)$.[1,2]

Definition 6. *(Gaussian-Bernoulli RBM) A GBRBM is a RBM for which:*

- $\boldsymbol{v}$ *takes values from* $\mathbf{R}^m$
- $\boldsymbol{h}$ *takes values from* $\{0,1\}^n$
- *Conditional probability of hidden given visible layer is defined same as in Definition 5.*

[1] σ is a sigmoid function defined with $\sigma(x) = (1 + e^{-x})^{-1}$
[2] $\mathcal{B}(\mu)$ denotes Bernoulli distribution with mean μ

- *Conditional expectation of visible given hidden layer is* $\mu^v = E[\boldsymbol{v}|\boldsymbol{h}] = b + W\boldsymbol{h}$ *and* $p(v_i|\boldsymbol{h}) = \mathcal{N}(\mu_i^v, 1)$.[3]

In previous definitions we do not give the expressions for joint probability functions of BBRBM and GBRBM, however they can be derived from given conditional probabilities. RBM is an energy based model and its joint distribution is defined through an energy function: function that is mapping each possible configuration of stochastic variables (visible and hidden) to scalar energy. In case of BBRBM the energy function is: $E(\mathbf{v},\mathbf{h}) = b^T\mathbf{v} - c^T\mathbf{h} - \mathbf{v}^T W\mathbf{h}$, while for GBRBM it is: $E(\mathbf{v},\mathbf{h}) = \frac{1}{2}(\mathbf{v}-b)^T(\mathbf{v}-b) - c^T\mathbf{h} - \mathbf{v}^T W\mathbf{h}$. The joint probability function is then $p(\mathbf{v},\mathbf{h}) = e^{-E(\mathbf{v},\mathbf{h})}/Z$, where Z represents a normalization constant, also called partition function. It is ensuring that total probability for all possible configurations (states) of visible and hidden units is 1 and $Z = \sum_{\mathbf{v}}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$ for BBRBM or $Z = \int_{\mathbf{v}}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$ for GBRBM. Calculating Z and exact probability (considering that evaluation of Z is required for calculating probability) for specific configuration is intractable in practice. This is because the sums (sum and integral) in expression for Z run through all possible configurations.

Notice that expressions for conditional expectations in Definitions 5 and 6 are defined over $\mathbf{R}^m(\mathbf{R}^n)$. This enables us to expand domain of binary unit to $[0,1]$ (instead of $\{0,1\}$) when we calculate expectations of other units condition on it—the value from $(0,1)$ is than interpreted as probability of that unit having value 1.

Gibbs sampling

It is possible to sample from a joint distribution modeled by RBM, which makes it generative model. A procedure that is often used for generating samples is Gibbs sampling (lines 1-4 in Algorithm 1). It is an iterative Markov chain Monte Carlo procedure that in each iteration samples each stochastic unit conditioned on the latest values of other units—this makes it efficient for RBM, since sampling all units of one layer (given another) can be done at once. These samples constitute a Markov chain, whose stationary distribution corresponds to the modeled joint distribution. Performing more steps (iterations) leads to more precise samples.

Training RBM

Training RBM is a process of learning weights and biases of RBM so it models some data distribution. Simplified, the training algorithm should adjust parameters (W, b, c from Definition 4) of a model so that probability for examples from the training set increases. RBM is trained using stochastic steepest ascent for maximizing the log likelihood of training data under the model, since finding the parameters analytically is not generally possible.

Contrastive divergence gives us an update rule for training RBM using stochastic steepest ascent. It is approximating derivative of the log likelihood of training set with respect to the parameters (W, b or c). The result of this approximation is very simple rule (Algorithm 1) for updating parameters of RBM where input example is presented to the visible layer of RBM and the parameters are updated in order to increase the probability of that example. Contrastive divergence is denoted CD-k, where k is the number of Gibbs steps used.

Previously described training procedure allows introduction of several modifications/additions to improve the learning process and the final model. Steepest ascent alows us to configure: batch processing of training examples, the number of epochs/iterations used. Contrastive divergence rule can be modified by: introduction of momentum, sparsity of hidden units, weight decay. It is also important to select proper type of RBM, number of hidden units, initial weights and biases. [11]

[3] $\mathcal{N}(\mu, 1)$ denotes Gaussian distr. with mean μ and variance 1

Algorithm 1 Contrastive divergence (CD-k) update rule, $k \in \mathbf{N}$, learning rate $\epsilon \in \mathbf{R}^+$

```
   In1: $\mathcal{RBM} = (W, b, c, p(\mathbf{h}|\mathbf{v}), p(\mathbf{v}|\mathbf{h}))$ from Definition 4
   In2: Training input $v_0 \in \mathbf{R}^m, m \in \mathbf{N}$
   Out: Parameters $W$,$b$ and $c$ of $\mathcal{RBM}$ are updated
1: for $i = 1...k$ do
2:     $h_{i-1} \leftarrow$ Sample from $p(\mathbf{h}|\mathbf{v} = v_{i-1})$ of $\mathcal{RBM}$
3:     $v_i \leftarrow$ Sample from $p(\mathbf{v}|\mathbf{h} = h_{i-1})$ of $\mathcal{RBM}$
4: end for
5: $h_0 \leftarrow E[\mathbf{h}|\mathbf{v} = v_0]$ of $\mathcal{RBM}$ (Definition 4)
6: $h_k \leftarrow E[\mathbf{h}|\mathbf{v} = v_k]$ of $\mathcal{RBM}$ (Definition 4)
7: $W \leftarrow W + \epsilon(v_0 h_0^T - v_k h_k^T)$
8: $b \leftarrow b + \epsilon(v_0 - v_k)$
9: $c \leftarrow c + \epsilon(h_0 - h_k)$
```

3.2 Deep belief net and greedy layer-wise training

This section describes deep belief net, a deep architecture used in this paper. The model and a learning algorithm are introduced in [12] and have been applied to many different tasks, some of which are classification, regression, natural language processing, information retrieval [13].

Deep belief net (DBN) can be used for finding low-dimensional representation of an input [14] by gradually extracting more and more abstract representation. These higher representations are expressed by activation of some subset of units (feature detectors) that are not mutually exclusive—they form a "distributed representation". Using DBN it is possible to extract binary features from raw continuous data (e.g. signal observations)—in our case we are interested in finding binary representations of reoccurring patterns in continuous signals from an industrial system. We want to find short binary representation (code) whose each unit (bit) captures some, possibly important, structure of the input data. Since each unit can have value of 0 or 1, it is separating input space on two regions. The whole code corresponds to some small region of the input space—all inputs from this region are "similar" from a perspective of DBN, as they activate the same units (feature detectors).

Example: Suppose we have a system with two drives used for movement of some objects left/right and up/down. Energy consumption of each of the drives is observed. Binary coding of these real-valued observations might give us 3 binary values whose interpretation could be: object is moved up/down, object is moved left/right and object is light/heavy (supposing that there are two different weight classes of the moved objects). It is usually not possible to interpret these individual binary values, except for very simple models.

DBN with $l \in \mathbf{N}$ hidden layers is a deep model whose top two layers ($\mathbf{h}_l$, $\mathbf{h}_{l-1}$) form a restricted Boltzmann machine and lower layers ($\mathbf{v}$,$\mathbf{h}_1$,...,$\mathbf{h}_{l-2}$) form a directed graphical model—stochastic neural network in which units in each layer are independent, given the values of the units in the layer above. DBN is illustrated in figure 2 and formally defined in Definition 7.

Definition 7. *(Deep belief net) A DBN with an architecture* m-n_1-...-n_l *models joint probability distribution*

$p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^l)$ *where* $\boldsymbol{v} = (v_1, ..., v_m)$ *is a vector (layer) of visible (input) units and* $\boldsymbol{h}^j = (h^j_1, ..., h^j_{n_j})|_{j \in \{1,...,l\}}$ *are* l *vectors (layers) of hidden units,* $m, n_1, ...n_l, l \in \mathbf{N}$. *It is a sequence* $\mathcal{DBN} = (\mathcal{RBM}_j)_{j \in \{1,...,l\}}$, *where:*

- $\mathcal{RBM}_j$ *is RBM from Definition 4*
- $p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^l) = p(\boldsymbol{h}^{l-1}, \boldsymbol{h}^l)(\prod_{k=1}^{l-2} p(\boldsymbol{h}^k|\boldsymbol{h}^{k+1}))p(\boldsymbol{v}|\boldsymbol{h}^1)$ *is a joint probability function*
- $p(\boldsymbol{h}^{l-1}, \boldsymbol{h}^l)$ *in equation for* $p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^l)$ *is joint probability function of* $\mathcal{RBM}_l$.
- $p(\boldsymbol{h}^j|\boldsymbol{h}^{j+1}))|_{j=1,...,l-2}$ *in equation for* $p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^l)$ *is conditional probability function of visible given hidden units of* $\mathcal{RBM}_j$ *from* $\mathcal{DBN}$.
- $p(\boldsymbol{v}|\boldsymbol{h}^1)$ *in equation for* $p(\boldsymbol{v}, \boldsymbol{h}^1, ..., \boldsymbol{h}^l)$ *is conditional probability function of visible given hidden units of* $\mathcal{RBM}_1$ *from* $\mathcal{DBN}$.
- $\boldsymbol{v}, \boldsymbol{h}^j|_{j \in \{1,...,l\}}$ *take values from the sets def. by* $\mathcal{RBM}_j$

Training DBN

In [12], greedy layer-wise training algorithm for training l-layer DBN from observations is presented, which sequentially trains l RBMs using contrastive divergence. The procedure is described in Algorithm 2. Each of the RBMs corresponds to one level of DBN. The input training set is used for training the first (lowest) RBM, while the training set for each next RBM is obtained by transforming the training set of the RBM below it, as described in Algorithm 2.

Algorithm 2 Training deep belief net, $l \in \mathbf{N}$ is number of hidden layers, meta parameters dbn_mp for training each RBM are provided

```
In: Training examples $\mathcal{X} = (x_i)_{i \in \{1,...,|\mathcal{X}|\}}$, where $|\mathcal{X}| \in \mathbf{N}$ is number of examples and $x_i \in \mathbf{R}^m$, $m \in \mathbf{N}$
Out: Learned model $\mathcal{DBN} = (\mathcal{RBM}_j)_{j \in \{1,...,l\}}$ (Def. 7)
1: $\mathcal{X}_1 \leftarrow \mathcal{X}$
2: for $j = 1, ..., l$ do
3:     $\mathcal{RBM}_j \leftarrow$ Train $\mathcal{RBM}_j$ using steepest ascent and CD-k (Algorithm 1) with training examples from $\mathcal{X}_j$
4:     $\mathcal{X}_{j+1} \leftarrow$ Transform each example $x_i$ from $\mathcal{X}_j$ using expectation of hidden given visible layer $E[\mathbf{h}|\mathbf{v} = x_i]$ of previously learned $\mathcal{RBM}_j$ (see Definitions 4, 5, 6)
5: end for
```

Theoretical justification of training algorithm and introduction of multiple levels in DBN is not subject of this paper. Refer to [12; 13] for more information.

Algorithm 3 DBN - Transformation of input vector to top level binary representation

```
In1: Input vector $v^1 \in \mathbf{R}^m, m \in \mathbf{N}$
In2: $\mathcal{DBN} = (\mathcal{RBM}_j)_{j \in \{1,...,l\}}$ (Definition 7)
Out: Binary code $b \in \{0,1\}^{n_l}, n_l \in \mathbf{N}$ is number of units at top layer of DBN
1: for $j = 1, ..., l$ do
2:     $h^j \leftarrow$ Expectation of hidden given visible layer of $\mathcal{RBM}_j$ from $\mathcal{DBN}$: $E[\mathbf{h}|\mathbf{v} = v^j]$ (see Definitions 4, 5, 6)
3:     $v^{j+1} \leftarrow h^j$
4: end for
5: $b \leftarrow$ Round each value from $h_l$ to nearest of {0,1}
```

Transforming input to top level binary representation

Modeling data distribution using DBN results in extracting higher level features from row input data [14]. Therefore, training DBN with Gaussian-Bernoulli RBM at the input layer and Bernoulli-Bernoulli RBMs at other layers enables us to transform real-valued input vector to top level binary representation. This procedure is described in Algorithm3.

Expected value of each unit at top layer of DBN ($\mathbf{h}_l$ in Algorithm 3) gives us the unit probability of having value 1. Each of these probabilities can be interpreted as intensity of input vector having some structure detected by corresponding unit. We round them to 0 or 1 to get binary representation of given input vector. Similarly, it is possible to propagate the binary code from the top layer back to the input layer. Hence, we are reconstructing the input from it's code.

4 Timed Automaton

The identification of behavior models for discrete signals requires different techniques than for continuous signals. A finite state automaton (FSA) is an established formalism which is able to capture the discrete behavior. Here, additionally the timing information is considered and therefore a timed automaton (TA) is identified.

Definition 8 (Timed Automaton)**.** *A timed automaton is a 4-tuple* $A = (S, \Sigma, T, \delta)$, *where*

- S *is a finite set of states. Each state* $s \in S$ *is a tuple* $s = (id, \boldsymbol{u})$, *where id is a current numbering and* $\boldsymbol{u}$ *is a discrete signal vector according to Definition 1.*
- Σ *is the alphabet, the set of events.*
- T *is a set of transitions. A transition is represented with* (s, a, δ, s'), *where* $s, s' \in S$ *are the source and destination states,* $a \in \Sigma$ *is the symbol and* δ *is a clock constraint. The automaton changes from state* s *to state* s' *triggered by a symbol* $a \in \Sigma$ *if the current clock value satisfies* δ. *The clock* c *is set to* 0 *after executing a transition, so that the clock starts counting time from executing this transition.*
- *A transition timing constraint* $\delta : T \to I$, *where* I *is a set of intervals.* δ *always refers to the time spent since the last event occurred. It is expressed as a time range or as a probability density function (PDF), i.e. as probability over time.*

The OTALA timed automaton learning algorithm uses the information from the signal vector to extract state information. The event between two states is created as the difference between the corresponding state vectors, additionally including the timing conditions (e.g. time range between minimum and maximum observation time or probability density function (PDF) over time). An outline of the algorithm OTALA is given in Algorithm 4, for a complete version refer to [15].

Simplified, the Algorithm 4 works as follows: For each incoming learning sample (vector of discrete signals including the time stamp), it is checked whether a corresponding state and transition exists (lines 3-5). If they already exist, they are updated (timing information, lines 5-12), otherwise they are created (lines 9-15). The identification runs until convergence (checked in lines 17-18).

Anomaly detection is done by comparing new observations with the behavior of the identified automaton. Generally, the behavior of a system can be described as a path through the automaton. As long as this path can be followed, the observed behavior corresponds to a normal behavior. Whereas, as soon as some observation cannot be mapped to the automaton, an anomaly is detected.

Algorithm 4 Timed automaton learning algorithm

```
    In: Observation example $D$ according to Definition 2
    Out: Timed automaton $\mathcal{A}$ according to Definition 8
 1: while (identificationConverged()==$false$) do
 2:     $t, \mathbf{u} \leftarrow$ getNextEvent($D$)
 3:     if stateExists($s, \mathbf{u}$) then
 4:         if transitionExists($s_{current}, s$) then
 5:             updateStateInformation($s$)
 6:             updateTransitionInformation($s_{current}, s, t$)
 7:         else
 8:             createNewTransition($s_{current}, s$)
 9:             updateStateInformation($s$)
10:         end if
11:     else
12:         createNewState($s, \mathbf{u}$)
13:     end if
14:     $s_{current} \leftarrow s$
15: end while
16: return $\mathcal{A}$
```

The anomaly detection procedure based on timed automata is first published in [16] and is outlined in Algorithm 5. The algorithm is based on the validation of incoming events. Therefore, in line 3 an event is extracted from the observation sample. The corresponding event is the difference between two subsequent signal vectors. In line 4, it is checked whether the event is available in the automaton, respectively, in line 5, the timing is checked. If an anomaly is found, it is returned in line 8 and 11 respectively, otherwise the anomaly detection is continued with the next state in line 6.

Algorithm 5 Anomaly Detection Algorithm

```
    In1: Timed automaton $\mathcal{A}$ (Definition 8)
    In2: Observation example $D$ according to Definition 2
    Out: Detected anomaly(if there exists one): $anomaly$
 1: $s \leftarrow s_0$, state is assigned to be the initial state $s_0$
 2: for all tuples in $D$ do
 3:     $e \leftarrow$extractEventFromObservation($D$)
 4:     if exists $e \in T$ then
 5:         if $t$ satisfies $\delta(T)$ then
 6:             $s \leftarrow s'$
 7:         else
 8:             return $anomaly$: wrong timing
 9:         end if
10:     else
11:         return $anomaly$: unknown event
12:     end if
13: end for
```

5 *DAD:DeepAnomalyDetection* algorithm

In this section we will outline the basic idea behind the proposed algorithm using a simple example and later we will formally define it.

5.1 Algorithm outline

DAD:DeepAnomalyDetection algorithm is outlined using artificial data set. In section 1, four steps of an algorithm are declared. Each of the steps will be described in more details using example data.

Example data set: We have generated a simple artificial time-series data set with four signals: three continuous and one discrete. Data set contains observations during 200 system cycles, obtained by modifying "base cycle". The sequence of this cycle is divided in six sub-sequences during each of which signals have constant values. Performed modifications of base signals include adding Gaussian noise to continuous signals and making small differences in duration of sub-sequences. One generated cycle is illustrated on left half of Figure 3.

Step 1: Windowing - One system cycle is time-series data over 30s at 150 time moments. The snapshots (Definition 3) are obtained using 3s window—a snapshot contains 15·3 real values (15 observations for each of the three continuous signals). Window overlapping of 30% was used which resulted in 2800 snapshots in deep learning training set.

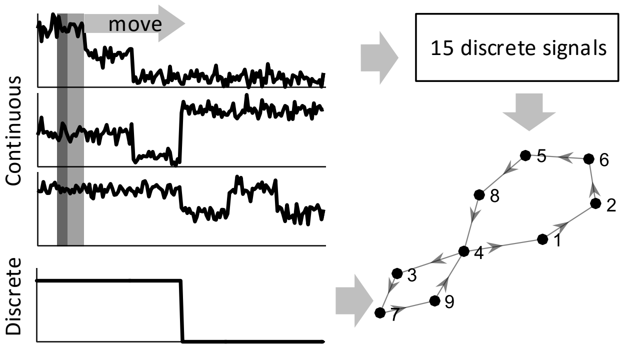

Figure 3: *DAD:DeepAnomalyDetection* applied on artificial data

Step 2: Deep Learning - In this step deep belief net is trained. Identifying the right meta parameters of DBN is usually done manually, based on experience, and it is not the subject of this paper. However, in this example we used four layer DBN with 15 units in top layer. Similar snapshots (according to DBN) are coded into the same binary value. Binary coding of all snapshots in one cycle and attaching information about the time stamp to each code is "discretizing" the signals, so the result is that we have replaced 3 continuous with 15 discrete signal. Notice that snapshots contain information about signals during 3s and each contains 45 real values.

Step 3: Learning Timed Automaton - 15 discrete signals that are result of transformation from continuous signals and one original discrete signal are merged together and used to learn timed automaton. The resulting automaton had nine states and its state diagram is shown in Figure 3.

Step 4 Anomaly detection - Anomaly detection in a sequence of new observations requires continuous observations to be discretized using DBN. The system behavior is then described by change of these discrete signals in time. When these signals are committed to the learned automaton—the path through the automaton is defining system behavior. As long as a path through the automaton can be followed, the observed behavior corresponds to normal behavior. Conversely, as soon as some combination of discrete signals cannot be mapped to the automaton, an anomaly is detected.

Anomalies are detected in one of the situations:

- Unknown event in the automaton caused by new pattern of continuous signals.
- Unknown event in the automaton caused by novel combination of discrete signals (original and transformed)
- Wrong timing of an event

- Unknown event caused by novel transition between two learned states

5.2 Formal definition of a new algorithm

Formally we will define *DAD:DeepAnomalyDetection* using two procedures: *Learning behavioral model* (Algorithm 6 and *Anomaly detection* (Algorithm 7). The first one includes: *Windowing*, *Deep learning* and *Learning automaton* steps while the second one includes *Anomaly detection* step. Outline of these steps is given in previous section.

Algorithm 6 Learning behavioral model, parameters are: window size $T_w \in \mathbf{R}_{\geq 0}$, windows overlap $T_o \in \mathbf{R}_{\geq 0}$, meta parameters for DBN training *dbn_mp*

```
In: Observations $\mathcal{O} = \{O_1, ..., O_{|\mathcal{O}|}\}, |\mathcal{O}| \in \mathbf{N}$ where $O_i|_{i\in\{1,...,|\mathcal{O}|\}}$ is an observation example (Definition 2)
Out1: $\mathcal{DBN} = (\mathcal{RBM}_j)_{j\in\{1,...,l\}}, l \in \mathbf{N}$ (Definition 7)
Out2: Timed automaton $\mathcal{A}$ (Definition 8)

1: for all $O_i \in \mathcal{O}$ do
2:     $S_i \leftarrow$ Apply windowing to continuous signals from $O_i$ using rectangular window of length $T_w$ and overlap $T_o$, $S_i = \{(t_k, s_k)|_{k\in\{1,...,|S_i|\}}\}$ where $s_k$ is a snapshot obtained at moment $t_k$, and $|S_i|$ is the number of obtained snapshots
3: end for
4: $\mathcal{DBN} \leftarrow$ Learn deep belief (Algorithm 2) using sequence $S = (s)|_{(\cdot,s)\in S_i, i=1,...,|S_i|}$ as input and $dbn\_mp$
5: for all $S_i \in \mathcal{S}$ do
6:     $D_i \leftarrow$ Transform continuous signals from $S_i$ (Algorithm 3) using $\mathcal{DBN}$ and $S_i$ as inputs and append result with original discrete signals from $O_i$
7: end for
8: $\mathcal{A} \leftarrow$ Learn automaton (Algorithm 4) using each of observation examples $D_1, ..., D_{|D|}, |D| \in \mathbf{N}$ as inputs
```

In Algorithm 6 first windowing is performed on each production cycle—observation example (Definition 2). This results in data set of snapshots for training deep belief net. DBN is trained using snapshots from all cycles. Learned DBN model enables us transformation of snapshots to binary codes which is used to transform continuous signals from each cycle to discrete signals: value of some k-th discrete signal at moment t_1 is value of the k-th bit of a binary code of the snapshot covering t_1. These discretized signals, together with original discrete signals are used for learning timed automaton.

Algorithm 7 Anomaly detection, parameters are: window size $T_w \in \mathbf{R}_{\geq 0}$, windows overlap $T_o \in \mathbf{R}_{\geq 0}$

```
In1: Observation example $O$ (Definition 2)
In2: Deep belief net $\mathcal{DBN}$ (Definition 7)
In3: Timed automaton $\mathcal{A}$ (Definition 8)
Out: Anomaly (if there exist one) $a$

1: $S \leftarrow$ Apply windowing to continuous signals from $O$ using rectangular window of length $T_w$ and overlap $T_o$, $S = \{(t_k, s_k)|_{k\in\{1,...,|S|\}}\}$ where $s_k$ is a snapshot obtained at moment $t_k$, and $|S|$ is the number of obtained snapshots
2: $D \leftarrow$ Transform continuous signals from $S$ (Algorithm 3) using $\mathcal{DBN}$ and $S$ as inputs and append original discrete signals from $O$
3: $a \leftarrow$ Detect anomaly (Algorithm 5), inputs are $\mathcal{A}$ and $D$
```

Algorithm 7 describes a simple procedure. For a new observation example, first windowing is applied in the same way as in algorithm 6 (the same window size and overlap). The sequence of snapshots is than transformed to discrete signals that are together with original discrete signals processed using algorithm 5.

Discussion about *DAD:DeepAnomalyDetection*

Success of anomaly detection phase depends highly of deep belief net model:

- If DBN does not generalize well, unseen snapshots from the normal behavior will be coded as wrong patterns and false anomaly will be detected.
- How is model separating input space on different regions (codes) is not predictable. Binary representation (code) could miss some signal structure that is important for distinguishing between two different patterns

We do not give any instructions for measuring quality of learned DBN model, except for using final accuracy score after applying *DAD:DeepAnomalyDetection* on validation data. DBN and timed automata are splitting the "job" here, therefore using different windowing and DBN configuration would result in different automaton. Number of top layer units of DBN, which is related to the number of states of automaton, probably plays most important role in configuring the algorithm. Effects of different configurations of the algorithm are not subject of this paper.

DBN is learning to represent well snapshots from the training set, however it is not predictable how are unseen anomalous snapshots going to be coded. They might be coded to a binary code that never occurred during normal behavior of the system which would automatically imply an anomaly. Another possible effect is that anomalous snapshot will be recognized as one of the patterns that are modeled in automaton, and then whether anomaly will be detected or not depends of learned automaton (transitions and timings). Detecting anomalous patterns is a side effect of the algorithm as it is not intended do detect this type of anomalies. The targeted anomalies are those with wrong timings of events in the automaton and transition to wrong state. If we assume that any new sequence off snapshots does not contain novel snapshots (patterns), then each snapshot will be coded as proper binary code and automaton will be responsible to detect anomaly—which is done in more predictable way.

6 Experiments

6.1 High rack storage system

High rack storage system is a demonstrator system for transportation and storage of objects/items. 13 continuous signals are observed in 13065 points during 48 operating cycles. The signals during one cycle are plotted on Figure 4 and they include measured and desired speed of several conveyors and their total energy consumption.

Deep learning data set was created without windowing, so each training example contained 12 real values (one of the signals was removed because it was constant). Each of the signals is normalized to unit variance. Deep belief net with 3 hidden layers was trained: 12-100-70-40 (the numbers represent number of units at each layer). All the examples from the training set were recognized as one of the 73 different patterns. The model enabled transformation of 12 continuous into 40 discrete signals, that were used for learning (non timed) automaton. State diagram of this automaton is illustrated on Figure 5.

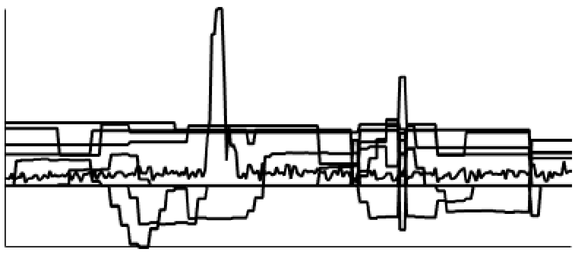

Figure 4: 13 continuous signals recorded from High Rack Storage System during one cycle. Signals are normalized to unit variance and ploted on the same axis.

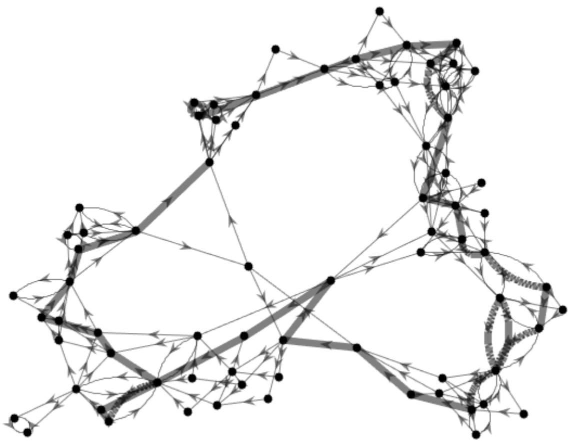

Figure 5: State diagram of the automaton learned from storage system data. Transitions during one of 48 cycles are presented with bold edges.

Figure 6 shows state diagram of the automaton (top of the figure) and signals in time during a part of one production cycle. After visual analysis of plots like this one, during several cycles, we believe that the model really identifies different patterns in input signals. E.g. in figure 6 there is a peak in top signal—during this peak automaton is in state 23. If the peak is too steep we could get transition from state 28 to 23 which would be detected as an anomaly - unexpected event.

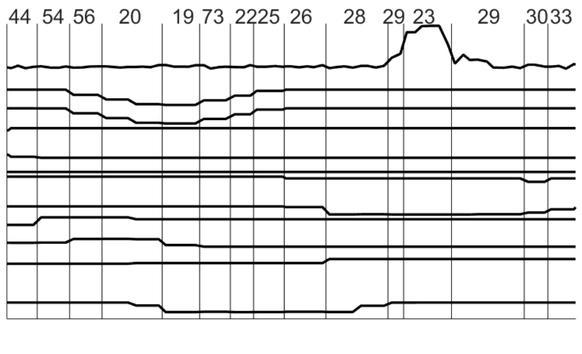

Figure 6: Change of signals and automaton states in time.

In order to test the algorithm we have artificially modified the signals by adding Gaussian noise with variance 1 at to each of them at some time moment. Table 1 shows how are different modifications recognized by *DAD:DeepAnomalyDetection*: as normal behavior, unexpected initial state, new pattern or unknown event.

Table 1: Anomaly detection in artificially modified data from Storage system

	Results of anomaly detection			
Modif.	Unexp. init. st.	New pattern	Unknown event	Normal
◊	44%	40%	1%	15%
□	0%	63%	24%	13%

◊ Added noise to observations at first time moment in a cycle
□ Added noise to observations at random time moment in a cycle

6.2 ATM card reader

In our second experiment we have used observations of an energy signal of an ATM card reader device/system that is consisted of several energy consuming devices some of which are: electrical motors and electromagnets. Data set contains measurements in 2311629 points during 250 operating cycles.

Deep learning data set was created using window size of 100 samples and 30% overlap, so each training example contained 100 real values. Deep belief net with 2 hidden layers was trained: 100-60-20. All examples from the training set were recognized as one of the 57 different patterns. The model enabled transformation of 1 continuous (latest 100 observations) into 20 discrete signals, that were used for learning (non timed) automaton.

Figure 7 shows energy consumption signal and change of the state of the automaton (printed above the signal) in time during a part of one operating cycle. Parts of the cycle in which there is no change of the state are cut from the plot. In addition, the figure also shows the reconstructions of the input snapshots (obtained by propagating the binary code back to the input layer). Figure 8 shows patterns learned by deep belief net. After visual analysis of plots like those on Figures 7 and 8, during several cycles, we believe that the model really identifies different patterns in input signal. E.g. in Figure 7 state 28 is followed by specific patterns recognized as state 5 and than 4. If these patterns are missing (the transition is from 28 to 27) it would be detected as an anomaly - unexpected event. If timed automaton is learned, staying too long in state 28 could be easily detected.

However, plot of pattern 5 in Figure 8 shows that different input patterns (classified by human) are sometimes recognized as the same pattern by DBN (pattern 5). This is probably because of the small number of such snapshots in DBN training set and it would result in algorithm failing to detect some anomalies.

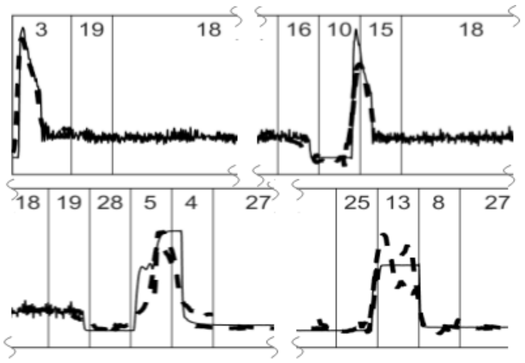

Figure 7: Change of states through time. Part of one cycle of Card Reader energy consumption is shown. Dashed line is reconstruction of snapshot (window overlap of 30% is used)

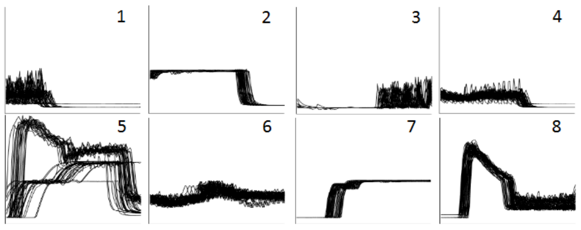

Figure 8: 8/57 learned patterns from card reader data—snapshots that are recognized as same discrete pattern are plotted on the same axis.

Table 2: Anomaly detection in artificially modified data from Card reader system

	Results of anomaly detection		
Modification	New pattern	Unknown event	Normal
◊	50%	45%	5%
□	36%	53%	11%
△	15%	35%	50%

◊ Signal dropped to zero
□ Signal raised by 50%
△ Added ramp raising from 0 to the variance of the signal

In order to test the algorithm we have artificially modified the energy consumption signal by modifying 100 consecutive measurements. Table 1 shows how are different modifications recognized by *DAD:DeepAnomalyDetection*: as normal behavior, new pattern or unknown event.

7 Conclusion

We have presented a new algorithm that tries to solve the problem of anomaly detection in hybrid production systems. The new algorithm *DAD:DeepAnomalyDetection* relies on deep belief nets and timed automata and minimizes requirements for expert knowledge. The sub procedures for creating behavioral model from observations and comparison of learned model with new observations are given formally. The algorithm was later tested using several data sets. The main purpose of the experiments was to visualize modeled behavior and show different types/classes of detected artificially created anomalies. Many tasks are left for future work some of which are: analysis of an effect of different configuration of deep belief net on accuracy, testing the algorithm using real anomalies from the systems.

References

[1] E.A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369, 2008.

[2] Ragunathan (Raj) Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.

[3] Peter C. Evans and Marco Annunziata. Industrial internet: Pushing the boundaries of minds and machines. Technical report, GE, 2012.

[4] L. Christiansen, A. Fay, B. Opgenoorth, and J. Neidig. Improved diagnosis by combining structural and process knowledge. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, Sept 2011.

[5] Rolf Isermann. Model-based fault detection and diagnosis - status and applications. In *16th IFAC Symposium on Automatic Control in Aerospace*, St. Petersbug, Russia, 2004.

[6] Oliver Niggemann and Volker Lohweg. On the Diagnosis of Cyber-Physical Production Systems - State-of-the-Art and Research Agenda. In *In:Twenty-Ninth Conference on Artificial Intelligence (AAAI-15)*, 2015.

[7] Oliver Niggemann, Gautam Biswas, John S. Kinnebrew, Hamed Khorasgani, Soeren Volgmann, and Andreas Bunte. Data-driven monitoring of cyber-physical systems leveraging on big data and the internet-of-things for diagnosis and control. In *International Workshop on the Principles of Diagnosis (DX)*, Paris, France, Aug 2015.

[8] Oliver Niggemann, Benno Stein, Asmir Vodenčarević, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1083–1090, Toronto, Ontario, Canada, 2012.

[9] Stefan Windmann, Florian Jungbluth, and Oliver Niggemann. A HMM-Based Fault Detection Method for Piecewise Stationary Industrial Processes. In *IEEE International Conference on Emerging Technologies and Factory Automatio (ETFA 2015)*, 2015.

[10] Jens Eickmeyer, Peng Li, Omid Givehchi, and Oliver Niggemann. Data driven modeling for system-level condition monitoring on wind power plants. In *In: The 26th International Workshop on Principles of Diagnosis (DX-2015)*, 2015.

[11] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.

[12] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[13] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[14] Geoffrey E. Hinton and Ruslan Salakhutdinov. Discovering binary codes for documents by learning deep generative models. *topiCS*, 3(1):74–91, 2011.

[15] Alexander Maier. Online passive learning of timed automata for cyber-physical production systems. In *The 12th IEEE International Conference on Industrial Informatics (INDIN 2014)*. Porto Alegre, Brazil, Jul 2014.

[16] A. Vodenčarević, H. Kleine Büning, O. Niggemann, and A. Maier. Using behavior models for anomaly detection in hybrid systems. In *Proceedings of the 23rd International Symposium on Information, Communication and Automation Technologies-ICAT 2011*, 2011.