Proceedings of the 2009 IEEE Symposium on Computational Intelligence
in Security and Defense Applications (CISDA 2009)

A Comparison of Techniques for On-line Incremental Learning of HMM Parameters in Anomaly Detection

Wael Khreich, Eric Granger, Ali Miri and Robert Sabourin

***Abstract*—Hidden Markov Models (HMMs) have been shown to provide a high level performance for detecting anomalies in intrusion detection systems. Since incomplete training data is always employed in practice, and environments being monitored are susceptible to changes, a system for anomaly detection should update its HMM parameters in response to new training data from the environment. Several techniques have been proposed in literature for on-line learning of HMM parameters. However, the theoretical convergence of these algorithms is based on an infinite stream of data for optimal performances. When learning sequences with a finite length, on-line incremental versions of these algorithms can improve discrimination by allowing for convergence over several training iterations. In this paper, the performance of these techniques is compared for learning new sequences of training data in host-based intrusion detection. The discrimination of HMMs trained with different techniques is assessed from data corresponding to sequences of system calls to the operating system kernel. In addition, the resource requirements are assessed through an analysis of time and memory complexity. Results suggest that the techniques for on-line incremental learning of HMM parameters can provide a higher level of discrimination than those for on-line learning, yet require significantly fewer resources than with batch training. On-line incremental learning techniques may provide a promising solution for adaptive intrusion detection systems.**

I. Introduction

Intrusion Detection Systems (IDSs) are used to identify, assess, and report unauthorized computer or network activities. Host-based IDSs (HIDSs) are designed to monitor the host system activities and state, while network-based IDSs monitor network traffic for multiple hosts. In either case, IDSs have been designed to perform misuse detection – looking for events that match patterns corresponding to known attacks – and anomaly detection – detecting significant deviations from normal system behavior.

Operating system events are usually monitored in HIDSs for anomaly detection. Since system calls are the gateway between user and kernel mode, early host-based anomaly detection systems monitor deviation in system call sequences [1]. Various detection techniques have been proposed to learn the normal process behavior through system call sequences [2]. Among these, techniques based on discrete Hidden Markov Models (HMMs) have been shown to shown to provide high level of performance [2].

Wael Khreich, Eric Granger and Robert Sabourin are in the Laboratoire d'imagerie, de vision et d'intelligence artificielle (LIVIA), École de technologie supérieure, Montreal, QC, Canada, (email: wael.khreich@livia.etsmtl.ca {eric.granger, robert.sabourin}@etsmtl.ca). Ali Miri is with the School of Information Technology and Engineering (SITE), and Department of Mathematics and Statistics, University of Ottawa, Ottawa, ON, Canada, (email:samiri@site.uottawa.ca).

HMM is stochastic process for sequential data [3]. Given an adequate amount of system call training data, HMM-based anomaly detectors can efficiently model the normal process behavior. A well trained HMM should be able to capture the underlying structure of the monitored application using the temporal order of system calls generated by the process. Once trained, an HMM provides a compact model, with tolerance to noise and uncertainty, which allows a fast evaluation during operations[1]. A normal sequence presented to HMM should produce a higher likelihood value than for a sequence that does not belong to the normal process pattern or language. Their ability to discriminate between normal and malicious sequences have been discussed in literature [4], [5], [6]. The effects on performance of the training set size, irregularity of the process, anomaly types, and number of hidden states of HMM were recently investigated in [7].

The primary advantage of anomaly-based IDS is the ability to detect novel attacks for which the signatures have not yet been extracted. However, anomaly detectors will typically generate false alarms mainly due to incomplete data for training, poor modeling, and difficulty in obtaining representative labeled data for validation. In practice, it is very difficult to acquire (collect and label) comprehensive data sets to design a HIDS for anomaly detection. Therefore, a major requirement for an anomaly detection system (ADS) is the ability to accommodate new data without the need to restart the training process with all accumulated data.

Most research found in literature for HMM-based anomaly detection using system calls assume being provided with a sufficient amount of data. Furthermore, the monitored process is not static – changes in the environment may occur, such as application update. This is also the case when fine tuning a base model to a specific host platform. Therefore, HMM parameters should be refined incrementally over time by accommodating newly acquired training data, to better fit the normal process behavior.

Standard techniques for training HMM parameters involve batch learning, based either on the Baum-Welch (BW) algorithm [8], a specialized expectation maximization (EM) technique [9], or on numerical optimization methods, such as the Gradient Descent (GD) algorithm [10]. Both approaches are iterative algorithms for maximizing the likelihood estimate

[1]In contrast, matching techniques that are based on look-up tables, e.g., STIDE (sequence time-delay embedding) must compare inputs to all normal training sequences. The number of comparisons increases exponentially with the detector window size DW, while for HMM evaluation the time complexity grows linearly with DW.

978-1-4244-3764-1/09/$25.00 ©2009 IEEE

(MLE) of the data. For a batch learning technique, the data sequence is assumed to be finite. Each training iteration of BW or GD involves observing all subsequences in the presented block[2] for training prior to updating HMM parameters. Successive iterations continues until some stopping criterion is achieved (e.g., likelihood drop on a validation set). Given a new block of data, an HMM trained with BW or GD must be trained from start using all cumulative training data.

As an alternative, on-line incremental algorithm updates the HMM parameters after each subsequence, yet is allowed to perform several iterations over all subsequences within the block (refer to Figure 1). Some desirable characteristics for on-line incremental learning include the ability to update HMM parameters from new training data, without requiring access to the previously-learned training data and without corrupting previously acquired knowledge [11].

In contrast, for an on-line learning technique, the data sequence is assumed to be infinite. Such techniques update HMM parameters after observing each subsequence, with no iterations. User-defined hyper-parameters remain constant or they are allowed to degrade monotonically over time. Given a new block, an HMM that performs on-line learning continues the training seamlessly. In practice when learning sequences with finite length, on-line learning may lead to poor performance.

Several techniques have been proposed in literature for different real-world applications. Among these, on-line learning techniques are based on the current sequence of observations for optimizing the objective function (commonly the MLE), and updating the HMM parameters. As with batch learning, they can also be divided into EM-based [12] and gradient-based [13], [14], [15] learning techniques. These on-line techniques are extended in this work to on-line incremental learning by allowing them to iterate over each block of data and by resetting the learning rates when a new block is presented. Another solution consists of learning an HMM for each new block of data then merging it with old ones using weight-averaging [16].

The objectives of this paper are to compare the techniques for on-line and on-line incremental learning of HMMs parameters when applied for anomaly HIDS application using system calls sequences. A synthetic generator of normal data plus injection of anomaly has been utilized to avoid various drawbacks encountered when experimenting with real data. The receiver operating characteristics (ROC) curves and the area under the ROC curve (AUC) are used as a measure of performance [17]. Analytical comparison of convergence time and resources requirements are also provided and discussed.

The rest of this paper is organized as follows. The next section discusses the importance of on-line incremental update for anomaly detection. Section 3 presents techniques for batch, on-line and on-line incremental learning for HMM parameters. The experimental methodology in Section 4 describes data generation, evaluation methods and performance metrics. Finally, simulation results are discussed in Section 5.

[2]A block of data is defined as a sequence of system call observations that has been segmented into overlapping subsequences according to a user-defined window size.

II. Incremental Learning in Anomaly Detection System

An crucial step to design an ADS is to acquire sufficient amount of data for training. In practice however, it is very difficult to collect, analyze and label comprehensive data sets due to many reasons that range from technical to ethical. In addition, even in the simplest scenario where no change in the environment is assumed, characterizing the sufficient amount of data required for building an efficient ADS is not a trivial task. In practice, limited data is always provided for training, and the computer environment is always susceptible to dynamic changes. This work focuses on providing solutions to the limited data problem as described with the following practical scenarios.

Given some amount of normal system call data for training, an ADS based on HMM could be trained, optimized and validated offline to provide some acceptable performance in terms of false and true positive rates. However, during operations, monitoring a centralized server for instance, the system is susceptible to produce a higher rate of false alarms than expected and tolerated by the system administrator. This is largely due to the limited data that is the provided for training. The anomaly detector will have a limited view of the normal process behavior, and rare events will be mistakenly considered as anomalous. Accordingly, the HMM detector should be refined, i.e., trained on some additional normal data when it becomes available, to better fit the normal behavior of process in consideration.

As a part of the detection system, the system administrator plays an important role for providing such new data. When an alarm is raised the suspicious system call subsequences are logged and the system administrator starts an investigation into other evidence of an attack. If an intrusion attempt is detected the response team will act to limit the damage, and the forensic analysis team try to find the cause of the successful attack. Otherwise, it is considered as a false alarm and the logged subsequences (which are possibly rare events) are tagged as normal and collected for updating the HMM detector. One challenge is the efficient integration of this newly-acquired data into the ADS without corrupting the existing knowledge structure, and thereby degrading the performance.

III. Techniques for Learning HMM Parameters

A discrete-time finite-state HMM is a stochastic process determined by the two interrelated mechanisms. A latent Markov chain having comprising N states in the finite-state space $S = \{S_1, S_2, ..., S_N\}$, and a set of observation discrete probability distributions $b_j(v)$, each one associated with a state [3], [18]. Starting from an initial state S_i, determined by the initial state probability distribution π_i, at each discrete-time instant, the process transits from state S_i to state S_j according to the transition probability distribution a_{ij} $(1 \leq i, j \leq N)$. The process then emits a symbol v according to the output

probability distribution $b_j(v)$ of the current state S_j. The model is therefore parametrized by the set $\lambda = (\pi, A, B)$, where vector $\pi = \{\pi_i\}$ is initial state probability distribution, matrix $A = \{a_{ij}\}$ denotes the state transition probability distribution, and matrix $B = \{b_j(k)\}$ is the state output probability distribution. Both A, B, and π' are row stochastic, which impose the following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \forall i, \sum_{k=1}^{M} b_j(k) = 1 \, \forall j, \text{ and } \sum_{i=1}^{N} \pi_i = 1 \quad (1)$$

$$a_{ij},\ b_j(k),\ \text{and } \pi_i \in [0,1],\ \forall ijk \quad (2)$$

A. Batch Learning

The target in HMM parameters learning is to train the model λ to best fit the observed batch of data $o_{1:T}$. The estimation of HMM parameters is frequently performed according to the maximum likelihood estimation (MLE) criterion[3]. MLE consists of maximizing the log-likelihood

$$\ell_T(\lambda) \triangleq \log \Pr(o_{1:T} \mid \lambda) \quad (3)$$

of the training data over HMM parameters space (Λ):

$$\lambda^* = \text{argmax}_{\lambda \in \Lambda} \ell_T(\lambda) \quad (4)$$

Unfortunately, since the log-likelihood depends on missing information (the latent states), there is no known analytical solution to the training problem. In practice, iterative optimization techniques (briefly described below) such as the Baum-Welch algorithm, a special case of the Expectation-Maximization (EM), or alternatively the standard numerical optimization methods such as gradient descent are usually used for this task.

In either case, the optimization requires the evaluation of the log-likelihood value (3) at each iteration, and the estimation of the conditional state densities. That is, the *smoothed a posteriori* conditional state density:

$$\gamma_t(i) \triangleq \Pr(q_t = i \mid o_{1:T}, \lambda) \quad (5)$$

and the *smoothed a posteriori* conditional joint state density:

$$\xi_t(i,j) \triangleq \Pr(q_t = i, q_{t+1} = j \mid o_{1:T}, \lambda) \quad (6)$$

The Forward-Backward (FB) [3] or the numerically more stable Forward-Filtering Backward-Smoothing (FFBS) [18] algorithms are typically used for computing the log-likelihood value (3) and the smoothed state densities of Eqs. (5) and (6).

The Baum-Welch (BW) algorithm [8] is an Expectation-Maximization (EM) algorithm [9] specialized for estimating HMM parameters. Instead of a direct maximization of the log-likelihood (3) BW optimizes the auxiliary $\mathcal{Q}$-function:

$$\mathcal{Q}_T(\lambda, \lambda^{(k)}) = \sum_{q \in S} \Pr(o_{1:T}, q_{1:T} \mid \lambda^{(k)}) \log \Pr(o_{1:T}, q_{1:T} \mid \lambda) \quad (7)$$

[3]Other criteria such as the maximum mutual information (MMI), and minimum discrimination information (MDI) could be also used for estimating HMM parameters. However, the widespread usage of the MLE for HMM is due to its attractive statistical properties – consistency and asymptotic normality – proved under quite general conditions.

which is the expected value of the complete-data log-likelihood and hence easier to be optimized. This is done by alternating between the expectation step (E-step) and maximization step (M-step). The E-step uses the FB or FFBS algorithms to compute the state densities (Eqs. 5 and 6) which are then used, in the M-step to re-estimate the model parameters:

$$\begin{aligned} \pi_i^{(k+1)} &= \gamma_1^{(k)}(i) \\ a_{ij}^{(k+1)} &= \frac{\sum_{t=1}^{T-1} \xi_t^{(k)}(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ b_j^{(k+1)}(m) &= \frac{\sum_{t=1}^{T} \gamma_t^{(k)}(j) \delta_{o_t v_m}}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned} \quad (8)$$

The Kronecker delta δ_{ij} is equal to one if $i = j$ and zero otherwise. Starting with an initial guess of the HMM parameters, λ^0, each iteration k, of the E- and M-step is guaranteed to increase the likelihood of the observations giving the new model until a convergence to a stationary point of the likelihood is reached [8].

In contrast, standard numerical optimization methods work directly with the log-likelihood function (3) and its derivatives. Starting with an initial guess of HMM parameters λ^0, the gradient descent (GD) updates the model at each iteration k using:

$$\lambda^{(k+1)} = \lambda^{(k)} + \eta_k \nabla_\lambda \ell_T(\lambda^{(k)}) \quad (9)$$

where the learning rate η_k could be fixed, or adjusted at each iteration. One way of computing the gradient of the likelihood $\nabla_\lambda \ell_T(\lambda^{(k)})$ is by using the values of the conditional densities (Eqs. 5 and 6) obtained from the FB or FFBS algorithms:

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{a_{ij}} \quad (10)$$

$$\frac{\partial \ell_T(\lambda^{(k)})}{\partial b_j(m)} = \frac{\sum_{t=1}^{T} \gamma_t(j) \delta_{o_t v_m}}{b_j(m)} \quad (11)$$

However, with the numerical optimization methods HMM parameters are not guaranteed to stay within their space limits. As described next, the parameters constraints (Eqs. 1 and 2) must therefore be imposed explicitly through a re-parametrization, to reduce the problem to unconstrained optimization.

Since, at each iteration, both BW and GD algorithms rely on the fixed-interval smoothing algorithms (FB or FFBS) to compute the state conditional densities, they are therefore performing a batch learning approach. This is because these fixed-interval smoothing algorithms require an access to the end of sequence in order to compute the smoothed densities. Similarly, when learning from a block of multiple subsequences, each iteration of the BW and GD requires the averaged smoothed densities over all the subsequences in the block. Therefore, the block must have a finite number of subsequences, and all the subsequences are visited at each iteration. Consequently, when provided with a new block of subsequences the training process must be restarted using the accumulated (old and new) data to accommodate the new data.

B. On-line and On-line Incremental Learning

Figure 1 presents an illustration of the batch, on-line, and on-line incremental learning approaches when provided with subsequent blocks each comprises R observation subsequences. When the first block (Ω_1) is presented, all algorithms start with the same initial guess of HMM parameters (λ_0). At each iteration k, batch algorithms update the model parameters using the averaged state densities (Eqs. 5 and 6) over all the subsequences in the block until stopping criteria are met, then the first operational model (λ_1) is produced. On-line algorithms directly update the model parameters using the stated densities based on each subsequence and output λ_1 upon reaching the last subsequence in the block. This constitutes one iteration of the on-line incremental algorithms which then re-iterate until reaching the stopping criteria before producing λ_1.

When the second block (Ω_2) is presented, batch algorithms restart the training procedure, using all accumulated training data ($\Omega_1 \cup \Omega_2$). The on-line algorithms however resumes training from the previous model (λ_1) using only the current block (Ω_2) without iterations, while the on-line incremental algorithms will re-iterate on Ω_2 until the stopping criteria are met.

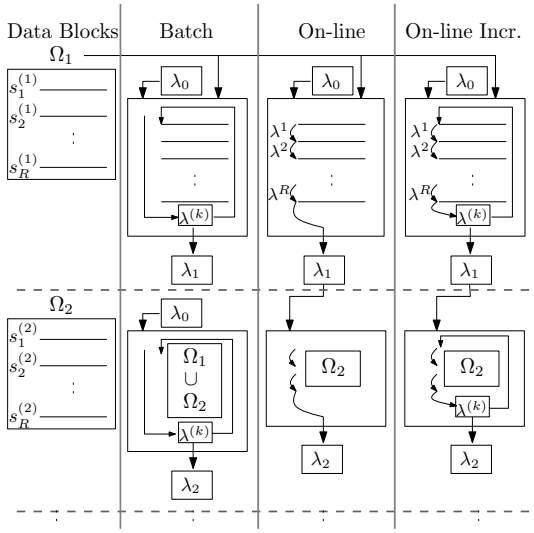

Fig. 1. Illustration of batch, on-line, and on-line incremental learning approaches when learning from subsequent blocks ($\Omega_1, \Omega_2, \ldots$) of R observation subsequences, provided at different time intervals.

On-line learning techniques for HMM parameters can be broadly divided into EM-based [12] or gradient-based [13], [15], [14] optimization of the log-likelihood function. These techniques are essentially derived from their batch counterparts, however the key difference is that the optimization and the HMM update are based on the currently presented subsequence of observations without iterations. EM-based techniques employ an indirect maximization of log-likelihood, through the complete log-likelihood (7), in which the E-step is performed on each subsequence of observation and then the model parameters are updated [12]. Gradient-based techniques however directly maximize the log-likelihood function (3) and then update HMM parameters after processing each subsequence of observation. Several gradient-based optimization algorithms have been proposed, such as the GD algorithm [13], the exponentiated gradient framework which also minimizes the model parameters divergence [14], and the recursive estimation technique [15].

The on-line learning technique proposed by Mizuno *et al.* is based on the BW algorithm, however applies a decayed accumulation of the state densities and a direct update of the model parameters after each subsequence of observations [12]. Starting with an initial model λ_0, the conditional state densities are recursively computed after processing each subsequence (r) of observations of length T by:

$$\sum_{t=1}^{T-1} \xi_t^{r+1}(i,j) = (1-\eta_r) \sum_{t=1}^{T-1} \xi_t^{r}(i,j) + \eta_r \sum_{t=1}^{T-1} \xi_t^{r+1}(i,j) \tag{12}$$

$$\sum_{t=1}^{T} \gamma_t^{r+1}(j)\delta_{o_t k} = (1-\eta_r) \sum_{t=1}^{T} \gamma_t^{r}(j)\delta_{o_t k} + \eta_r \sum_{t=1}^{T} \gamma_t^{r+1}(j)\delta_{o_t k} \tag{13}$$

and the model parameters are then directly updated using (8). The learning rate η_k is proposed in polynomial form $\eta_r = c(\frac{1}{r})^d$ for some positive constants c and d.

Based on the GD (9) of the negative likelihood, the on-line algorithm for HMM parameters estimations introduced by Baldi and Chauvin [13] employs a softmax parametrization to transform the constraint optimization into an unconstrained one. This is achieved by mapping the bounded space (a, b) to the unbounded space (u, v):

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_k e^{u_{ik}}} \text{ and } b_j(k) = \frac{e^{v_j(k)}}{\sum_z e^{v_j(z)}} \tag{14}$$

The transformed parameters are then updated, after each subsequence of observations, as follows:

$$u_{ij}^{r+1} = u_{ij}^{r} + \eta \sum_{t=1}^{T} \left(\xi_t^{r+1}(i,j) - a_{ij}\gamma_t^{r+1}(i) \right) \tag{15}$$

$$v_{j}^{r+1}(k) = v_{j}^{r}(k) + \eta \sum_{t=1}^{T} \left(\gamma_t^{r+1}(j)\delta_{o_t k} - b_j(k)\gamma_t^{r+1}(j) \right) \tag{16}$$

The objective function proposed by Singer *et al.* [14] minimizes the divergence between the old and new model parameters penalized by the negative log-likelihood of each subsequence multiplied by a fixed positive learning rate ($\eta > 0$):

$$\lambda^{r+1} = \arg\min_{\lambda} \left(KL(\lambda^{r+1}, \lambda^{r}) - \eta \ell_T(\lambda^{r+1}) \right) \tag{17}$$

The Kullback-Leibler (KL) divergence (or relative entropy) is defined between two probability distributions $P_{\lambda^{(k)}}(o_{1:t})$ and $P_{\lambda}(o_{1:t})$ by:

$$KL(P_{\lambda^r} \parallel P_{\lambda}) \quad = \quad \sum_t \Pr_{\lambda^r}(o_{1:t}) \log \frac{\Pr_{\lambda^r}(o_{1:t})}{\Pr_{\lambda}(o_{1:t})} \tag{18}$$

KL is always non-negative and attains its global minimum at zero for $\Pr_\lambda \rightarrow \Pr_{\lambda^r}$. This optimization is based on the exponentiated gradient framework, therefore the parameters constraints are respected implicitly. After processing each subsequence of observations, the model parameters are updated using the conditional state densities and the derivatives of the log-likelihood ((10) and (11)):

$$a_{ij}^{r+1} = \frac{1}{Z_1} a_{ij}^r e^{\left(-\frac{\eta}{\sum_{t=1}^T \gamma_t^{r+1}(i)} \frac{\partial \ell_T(\lambda^{r+1})}{\partial a_{ij}}\right)} \tag{19}$$

$$b_j^{r+1}(k) = \frac{1}{Z_2} b_j^r(k) e^{\left(-\frac{\eta}{\sum_{t=1}^T \gamma_t^{r+1}(j)} \frac{\partial \ell_T(\lambda^{r+1})}{\partial b_j(k)}\right)} \tag{20}$$

where Z_1 and Z_2 are normalization factors.

The idea proposed by Ryden [15] is to consider successive subsequences of T observations taken from a data stream, $\boldsymbol{o}_r = \{o_{(r-1)T+1}, \ldots, o_{rT}\}$, as independent of each other. This assumption reduces the extraction of information from all the previous observations to a data-segment of length T. In fact, this has been considered implicitly with all the above techniques that process multiple subsequences of T observations. To enforce parameters stochastic constraints (1), a projection ($\mathbb{P}_G$) on a simplex is suggested, which updates all but one of the parameters in each row of the matrices. At each iteration, the recursion is given (without matrix inversion)

$$\lambda^{r+1} = \mathbb{P}_G\left(\lambda^r + \eta_r h(\boldsymbol{o}_{r+1} \mid \lambda^r)\right) \tag{21}$$

where $h(\boldsymbol{o}_{r+1} \mid \lambda^r) = \nabla_{\lambda^r} \log \Pr(\boldsymbol{o}_{r+1} \mid \lambda^r)$ and $\eta_r = \eta_0 r^{-\rho}$ for some positive constant η_0 and $\rho \in (\frac{1}{2}, 1]$. It was shown to converge almost surely to the set of Kuhn–Tucker points for minimizing the Kullback-Leibler divergence $KL_k(\lambda^r \parallel \lambda^{(true)})$ defined in (18). KL attains its global minimum at $\lambda^r \rightarrow \lambda^{(true)}$, provided that the HMM is identifiable, therefore the subsequence must contains at least two symbols ($T \geq 2$).

On-line learning algorithms are therefore proposed for situations where a long (ideally infinite) sequences of observation are provided for training. In this paper however, these techniques are applied in an on-line incremental fashion, where algorithms are allowed to converge over several iterations of each new training block. However, when a new block is provided all learning rates are reset.

IV. Experimental Methodology

The University of New Mexico (UNM) data sets are commonly used for benchmarking ADS based on system calls sequences. Normal data are collected from a monitored process in a secured environment, while testing data are the collection of the system calls when this process is under attack [2]. Since it is very difficult to isolate the manifestation of an attack at the system call level, the UNM test sets are not labeled. Therefore, in related work, intrusive sequences are usually labeled in comparison with the normal subsequences, (e.g., using STIDE). This labeling process leads to a biased evaluation of techniques, which depends on both training data size and detector window size.

The need to overcome issues encountered when using real-world data for anomaly-based HIDS (incomplete data for training, and labeled data) has lead to the implementation of a synthetic data generation platform for proof-of-concept simulations. It is intended to provide normal data for training and labeled data (normal and anomalous) for testing. This is done by simulating different processes with various complexities then injecting anomalies in known locations.

Inspired by the work of Tan and Maxion [19], [20], the data generator is based on the Conditional Relative Entropy (CRE) of a source. It is defined as the conditional entropy divided by the maximum entropy (*MaxEnt*) of that source, which gives an irregularity index to the generated data. For two random variables x and y the CRE can be computed by:

$$CRE = \frac{-\sum_x p(x) \sum_y p(y \mid x) \log p(y \mid x)}{MaxEnt} \tag{22}$$

where for an alphabet of size Σ symbols, $MaxEnt = -\Sigma \log(1/\Sigma)$ is the entropy of a theoretical source in which all symbols are equiprobale. It normalizes the conditional entropy values between $CRE = 0$ (perfect regularity) and $CRE = 1$ (complete irregularity or random). In a subsequence of system calls, the conditional probability, $p(y \mid x)$, represents the probability of the next system call given the current one. It can be represented as the columns and rows (respectively) of a Markov Model with the transition matrix $M = \{a_{ij}\}$, where $a_{ij} = p(S_{t+1} = j \mid S_t = i)$ is the transition probability from state i at time t to state j at time $t + 1$. Accordingly, for a specific alphabet size Σ and CRE value, a Markov chain is first constructed, then used as a generative model for normal data. This Markov chain is also used for labeling injected anomalies as described below. Let an anomalous event be defined as a surprising event which does not belong to the process normal pattern. This type of event may be a *foreign-symbol* anomaly subsequence that contains symbols not included in the process normal alphabet, a *foreign n-gram* anomaly subsequence that contains *n-grams* not present in the process normal data, or a *rare n-gram* anomaly subsequence that contains *n-grams* that are infrequent in the process normal data and occurs in burst during the test[4].

Generating training data consists of constructing Markov transition matrices for an alphabet of size Σ symbols with the desired irregularity index (CRE) for the normal sequences. The normal data sequence with the desired length is then produced with the Markov chain, and segmented using a sliding window (shift one) of a fixed size, DW. To produce the anomalous data, a random sequence ($CRE = 1$) is generated, using the same alphabet size Σ, and segmented into subsequences of a desired length using a sliding window with a fixed size of AS. Then, the original generative Markov chain is used to compute the likelihood of each subsequence. If the likelihood is lower than a threshold it is labeled as anomaly. The threshold is set to $(\min(a_{ij}))^{AS-1}, \forall_{i,j}$, the minimal

[4]This is in contrast with other work which consider rare event as anomalies. Rare events are normal, however they may be suspicious if they occurs in high frequency over a short period of time.

value in the Markov transition matrix to the power $(AS-1)$, which is the number of symbol transitions in the subsequence of size AS. This ensures that the anomalous subsequences of size AS are not associated with the process normal behavior, and hence foreign *n-gram* anomalies are collected. The trivial case of foreign-symbol anomaly is disregarded since it is easy to be detected. Rare *n-gram* anomalies are not considered since we seek to investigate the performance at the detection level, and such kind of anomalies are accounted for at a higher level by computing the frequency of rare events over a local region. Finally, to create the testing data another normal sequence is generated, segmented and labeled as normal. The collected anomalies of the same length are then injected into those subsequences at random according to a mixing ratio.

In the presented experiments, a normal data sequence of length $1,600$ symbols is produced using a Markov model with an irregularity index $CRE = 0.4$, and segmented using a sliding window of a fixed size, $DW = 8$ [7]. The data are then divided into 10 blocks, Ω_i, for $i = 1, \ldots 10$, each comprises $R = 20$ subsequences. A test set of 400 subsequences each of size $AS = 8$ is prepared as described above. It comprises 75% of normal and 25% of anomalous data. Each block of the normal data Ω_i is divided into blocks of equal size – one is used for training (Ω_i^{train}) and the other for validation (Ω_i^{valid}), which is used to reduce the overfitting effects (hold-out validation).

For batch algorithms (BW-batch and GD-batch), successive blocks for training are accumulated and training is restarted each time a new block is presented. That is, the HMMs are first trained and validated on the first block of data (Ω_1^{train},Ω_1^{valid}). The training is then restarted, by re-initializing the models at random, and performing the batch algorithms using the accumulated blocks of data ($\Omega_1^{train} \cup \Omega_2^{train}$,$\Omega_1^{valid} \cup \Omega_2^{valid}$), and so on. In contrast, the on-line incremental algorithms resume training by starting with the corresponding models produced from the previous block and by using the presented block only ($\Omega_i^{train}, \Omega_i^{valid}$). The stopping criterion for the batch and the on-line incremental algorithms is set to a maximum of 100 iterations or to when the log-likelihood remains constant for at least 10 iterations for the validation data. On-line learning algorithms are not allowed to iterate on successive blocks of data. In this case, all learning rates are optimized and reset with the presentation of each new block. In all cases, the algorithms are applied to an ergodic (fully connected) HMM with eight hidden states ($N = 8$), and are initialized with the same random model. For each algorithm, the model that produced the highest log-likelihood value on the validation data is selected for testing.

The log-likelihood of the test data (normal + anomalous) are then evaluated using the forward algorithm. By sorting the test subsequences decreasing by these log-likelihood values, and updating the true positive rate (tpr) and the false positive rate (fpr) while moving down the list, results in a Receiver Operating Characteristics (ROC) curves [17]. The ROC curve depicts the trade-off between the tpr – the number of anomalous subsequences correctly detected over the total number of anomalous subsequences – and the fpr – the number of normal subsequences detected as anomalous over the total number of normal subsequences in the test set. The Area Under the ROC Curve (AUC) is used as a measure of performance. $AUC = 1$ means a perfect separation between normal and anomalous ($tpr = 100\%, fpr = 0\%$), while $AUC = 0.5$ means a random classification.

This procedure is replicated ten times with different training, validation and testing sets, and the resulting AUCs are averaged and presented along with their standard deviations (error bars). Although not show in this paper, various experiments have been conducted using different values of N, CRE, DW and Σ. These experiments produced similar results and hence the below discussion hold.

V. Results

The EM-based algorithm (Mizuno [12]), and the gradient-based ones (Baldi [13], Singer [14], and Ryden [15]) are first applied in an on-line learning approach as originally proposed by the authors. Figure 2 presents the average AUC achieved for on-line techniques for each block of training data. The results of the batch BW and GD algorithms are presented for reference.

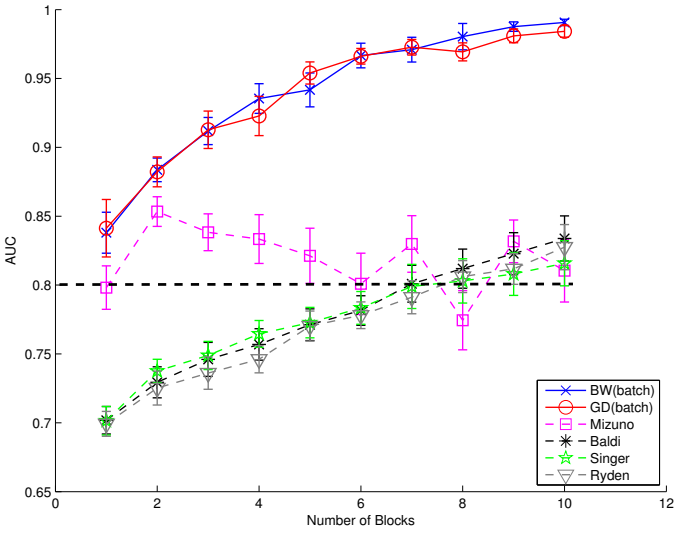

Fig. 2. Average AUC of on-line learning techniques vs the amount of training data that are used to train an ergodic HMM with $N = 8$.

The performances of the on-line algorithms tend to be equivalent with the increase of data. This is also confirmed by the statistical tests, conducted at the final block of data as shown in Figure 4. However, at the beginning, when few blocks of data are presented, the EM-based algorithm performs better than gradient-based ones. Due to one view of the data, on-line algorithms require a large amount of data to achieve good performance. Theoretically, an infinite amount of data is assumed when trying to prove the convergence of such on-line algorithms. Among the presented techniques, only Ryden provided convergence analysis and proof of consistency [15]. The average performances of the batch algorithms are

equivalent and increase with the number of blocks. This is expected, since the batch algorithms are allowed to iterate on the accumulated data, they have therefore a global (backward) view.

Figure 3 presents the averaged AUC achieved by on-line incremental techniques for each block of training data. The average AUC of the on-line incremental algorithms are lower than the performance of batch and higher than that of the on-line ones. In fact, these algorithms have a local view of the presented data. Although they are allowed to learn the new data through several iterations, there is a loss of information form the previously learned data. This is usually controlled with the decaying learning rate, which assigns a weight to the past information with reference to the future data contribution.

Among the on-line incremental algorithms, statistical testing shows that Baldi's algorithm [13] achieved slightly superior performance than the others as shown in Figure 4. This is possibly related to the short length of training subsequences ($DW = 8$). The other on-line incremental algorithms may require larger subsequences to accumulate enough information from each one before updating the model parameters. For instance, Ryden [15] suggests using a minimum subsequence length of $DW = 20$.

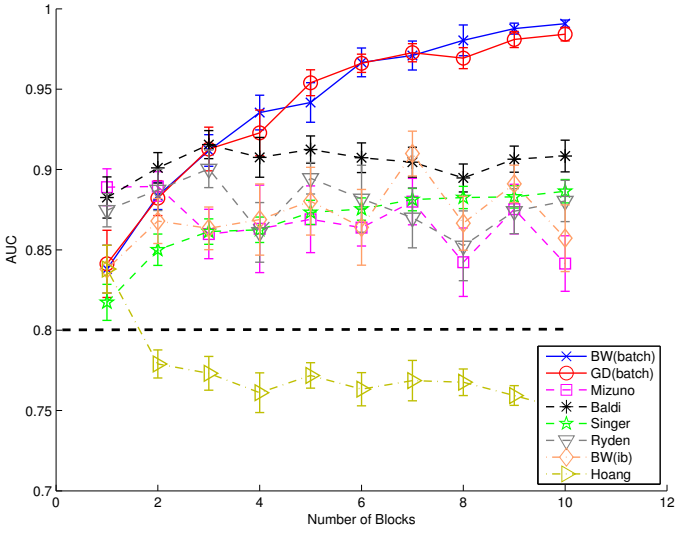

Fig. 3. Average AUC of on-line incremental learning techniques vs the amount of training data that are used to train an ergodic HMM with $N = 8$.

However, the stochastic nature of the on-line incremental algorithms allows them to escape local minima. This important characteristic can be shown when both batch and incremental algorithms are trained on the same data. For instance, this is illustrated when learning the first block in Figure 3. It can be seen that on-line incremental algorithms are capable of producing superior results to their batch counterparts. Therefore, they may be used as an alternative for batch training, especially that they converge faster than the batch algorithms since they update the model after each subsequence and hence exploit new information faster.

Figure 3 also presents the results of the BW incremental batch, BW(ib) for reference. That is, when learning Ω_1, BW is initialized with a random HMM and the algorithm is applied until it converges. For the subsequent block Ω_2, BW is then initialized with λ_1. The performance of BW(ib) indicates data corruption since it is prone to get stuck in a local minimum from the previous block. Interestingly, this straightforward EM-based solutions produced similar results to Mizuno [12]. This indicates that the learning rates employed by the latter during the experiments may be better optimized to escape local minima.

In addition, Figure 3 includes another incremental approach based on learning an HMM for each new block of data then merging it with old ones using weight-averaging [16]. This learn and merge approach performed statistically worse than most of the on-line incremental techniques as shown in Figure 4. This is due to averaging ergodic HMMs since the states order may be mixed up between the HMM trained on the first block and that trained on the second block of data.

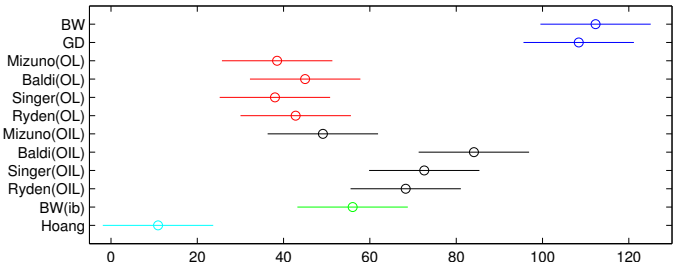

Fig. 4. Kruskall-Wallis (one-way analysis of variance) statistical test for batch, on-line (OL) and on-line incremental learning (OIL) algorithms after processing the final block Ω_{10} of data.

Table I compares the time and memory complexity of EM-based and gradient-based algorithms each processing a subsequence of T observations and then updating the model parameters. This represents the core operations required by all learning approaches. Time complexity represents the worst-case number of operations required for one iteration of EM-based and gradient-based algorithms. For both algorithms one iteration involves computing one forward and one backward pass with $o_{1:T}$. Memory complexity is the worst-case number of 32 bit words needed by the algorithms to store the required temporary variables in RAM. Since both algorithms rely on the FB or FFBS to compute the state conditional densities (Eqs. 5 and 6), therefore they require about the same computational time complexity, $\mathcal{O}(N^2T)$, and the same memory requirements, $\mathcal{O}(NT)$. The additional computation time required by the gradient-based algorithms while updating HMM parameters stems from the re-parametrization (Section 3).

For a block of R subsequences, batch learning involves R times the computations of the state densities whereas only one update of the parameters is performed at each iteration. On the other hand, the on-line algorithms perform R times the computation of state densities and R times the update of the model, without iterating however. On-line algorithms are therefore the fastest in learning HMM parameters. At each iteration, the on-line incremental algorithms require the same

TABLE I

WORST-CASE TIME AND MEMORY COMPLEXITY ANALYSIS FOR EM- AND GRADIENT-BASED ALGORITHMS EACH PROCESSING A SUBSEQUENCE OF T OBSERVATIONS, WITH AN N STATE HMM STATES AND AN ALPHABET OF SIZE $\Sigma = M$ SYMBOLS. THIS REPRESENTS THE CORE OPERATIONS REQUIRED BY BATCH, ON-LINE AND ON-LINE INCREMENTAL ALGORITHMS, THE DIFFERENCES STEM FROM ITERATING UNTIL CONVERGENCE.

Algorithms	Estimation	Time			Memory
		# Multiplications	# Divisions	# Exponentiation	
EM-based	State Prob. (Eqs. 5 & 6)	$6N^2T + 3NT - 6N^2 - N$	$N^2T + 3NT - N^2 - N$		$NT + N^2 + 2N$
	Transition Prob. (A)	N^2	N^2		N^2
	Emission Prob. (B)	NM	NM		NM
	Total	$6N^2T + 3NT - 5N^2 + NM - N$	$N^2T + 3NT + MN - N$		$NT + 2N^2 + NM + 2N$
Gradient-based	State Prob. (Eqs. 5 & 6)	$6N^2T + 3NT - 6N^2 - N$	$N^2T + 3NT - N^2 - N$		$NT + N^2 + 2N$
	Transition Prob. (A)	N^2	N^2	N^2	N^2
	Emission Prob. (B)	NM	MN	NM	NM
	Total	$6N^2T + 3NT - 5N^2 + NM - N$	$N^2T + 3NT + MN - N$	$N^2 + NM$	$NT + 2N^2 + NM + 2N$

computational time need by the on-line ones. Accordingly, the on-line incremental techniques require more computational time per iteration than the batch counterparts, however fewer iterations are required to converge. The memory complexity of the on-line algorithms is constant in time, $\mathcal{O}(NT)$, and also for the on-line incremental ones though it is R times greater. However, for batch algorithms this value scales linearly with the number of the accumulated subsequences.

VI. CONCLUSION

In this paper, the performance of several techniques is compared for on-line incremental learning of HMM parameters as new sequences of training data becomes available. These techniques are considered for updating host-based intrusion detection systems from system calls to the OS kernel, but apply to other HMM-based detection systems that face the challenges associated with limited training data and environmental changes. Results have shown that techniques for on-line incremental learning of HMM parameters can provide a higher level of discrimination than those for on-line learning, yet require significantly fewer resources than with batch training. On-line incremental learning techniques may provide a promising solution for adaptive intrusion detection systems. Future work includes in-depth investigation of the learning rates effects on performances, which are sensitive parameters especially for gradient-based techniques. Comparing these techniques with models combinations at the training level or responses fusion at the decision level is also an interesting direction to explore.

REFERENCES

[1] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in *Proc. of the 1996 IEEE Symp. on Research in Security and Privacy*, 1996, pp. 120–128.

[2] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proc. of the IEEE Computer Society Symp. on Research in Security and Privacy*, 1999, pp. 133–45.

[3] L. Rabiner, "A tutorial on HMM and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[4] B. Gao, H.-Y. Ma, and Y.-H. Yang, "HMMs based on anomaly intrusion detection method," *Proc. of 2002 Int'l Conf. on Machine Learning and Cybernetics*, vol. 1, pp. 381–385, 2002.

[5] X. D. Hoang, J. Hu, and P. Bertok, "A multi-layer model for anomaly intrusion detection," in *IEEE Int'l Conf. on Networks.*, vol. 1, 2003, pp. 531–536.

[6] W. Wang, X.-H. Guan, and X.-L. Zhang, "Modeling program behaviors by HMMs for intrusion detection," *Proc. of 2004 Int'l Conf. on Machine Learning and Cybernetics*, vol. 5, pp. 2830–2835, 2004.

[7] W. Khreich, E. Granger, R. Sabourin, and A. Miri, "Combining hidden markov models for anomaly detection," in *International Conference on Communications (ICC)*, Dresden, Germany, 2009.

[8] L. E. Baum, G. S. Petrie, and N. Weiss, "A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Stat.*, vol. 41, no. 1, pp. 164–171, 1970.

[9] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood estimation from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[10] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *Bell System Tech. J.*, vol. 62, pp. 1035–1074, 1983.

[11] R. Polikar, L. Upda, S. Upda, and V. Honavar, "Learn++: an incremental learning algorithm for supervised neural networks," *IEEE Trans. on Systems, Man and Cybernetics (C)*, vol. 31, no. 4, pp. 497–508, 2001.

[12] J. Mizuno, T. Watanabe, K. Ueki, K. Amano, E. Takimoto, and A. Maruoka, "On-line estimation of hidden Markov model parameters," *Proc. of 3rd Int'l Conf. Discovery Science*, vol. 1967, pp. 155–69, 2000.

[13] P. Baldi and Y. Chauvin, "Smooth on-line learning algorithms for hidden Markov models," *Neural Computation*, vol. 6, no. 2, pp. 307–318, 1994.

[14] Y. Singer and M. K. Warmuth, "Training algorithms for hidden markov models using entropy based distance functions." in *NIPS*, 1996, pp. 641–647.

[15] T. Ryden, "Asymptotic efficient recursive estimation for incomplete data models using the observed information," *Metrika*, vol. 44, pp. 119–145, 1998.

[16] X. Hoang and J. Hu, "An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls," in *IEEE Int'l Conf. on Networks.*, vol. 2, 2004, pp. 470–474.

[17] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letter.*, vol. 27, no. 8, pp. 861–874, 2006.

[18] Y. Ephraim and N. Merhav, "Hidden markov processes," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1518–1569, 2002.

[19] R. Maxion and K. Tan, "Benchmarking anomaly-based detection systems," in *Proc. of the 2000 Int'l Conf. on Dependable Systems and Networks*, 2000, pp. 623–630.

[20] K. Tan and R. Maxion, "Determining the operational limits of an anomaly-based intrusion detector," *IEEE J. on Selected Areas in Communications*, vol. 21, no. 1, pp. 96–110, 2003.