Malware Identification with Dictionary Learning

Paul Irofti*†, Andra Băltoiu†
* *Department of Computer Science*, *University of Bucharest*, Romania
† *The Research Institute of the University of Bucharest (ICUB)*, *University of Bucharest*, Romania
Emails: paul@irofti.net, andra.baltoiu@fmi.unibuc.ro

***Abstract*—Malware identification is a difficult task that has been recently approached by training classifiers through machine learning. We present here a low complexity semi-supervised dictionary learning framework that begins with training an initial dictionary on a small labeled data set, and then continues with online learning on incoming unlabeled data, making use of every sample that it is exposed to, with the scope of adapting to new and unknown malware types. Our main contribution is a new online algorithm that makes use of regularization techniques that balance the capability of the dictionary to express both fresh and well established patterns.**

***Index Terms*—malware identification, online semisupervised learning, dictionary learning, sparse representations**

I. Introduction

The spread of networking-capable daily-usage embedded devices has lead to the phenomenon known as the Internet of Things (IoT), where these low-power machines provide intelligent services by calling out to high-performance servers to process their data through machine learning (ML) algorithms. This externalization of data raises privacy and security concerns [1], but also hinders the adaptation and, possibly, the retraining of the ML models on the particular data that each IoT device is exposed to. In this paper we address these problems by creating a light-weight ML framework based on dictionary learning (DL) with sparse representations [2].

The dictionary learning problem starts with a set of N training signals of length m grouped as the columns of the matrix $\boldsymbol{Y} \in \mathbb{R}^{m \times N}$ and attempts to solve

$$\begin{aligned} \min_{\boldsymbol{D},\boldsymbol{X}} \quad & \|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}\|_F^2 \\ \text{s.t.} \quad & \|\boldsymbol{x}_\ell\|_0 \leq s,\ \ell = 1:N \\ & \|\boldsymbol{d}_j\| = 1,\ j = 1:n \end{aligned} \tag{1}$$

in search of the optimal dictionary $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ producing s-sparse representations $\boldsymbol{X} \in \mathbb{R}^{n \times N}$ such that their product is a close approximation of $\boldsymbol{Y}$. The dictionary columns, also called atoms, are normalized to avoid indetermination due to the $\boldsymbol{DX}$ product. Existing methods approach the problem through alternate optimization strategies, where first the dictionary is fixed and the representations are computed, and last the representations are fixed and the dictionary is updated. This process is iterated multiple times as needed. For both stages fast and efficient algorithms exist that provide state of the art results in many applications [2]. We mention the fundamental algorithms OMP [3] for the representation stage, and K-SVD [4] and MOD [5] for dictionary update.

In the following we study the application of DL to the malware identification problem. We view this as a binary-classification problem, where the DL algorithm instills discriminatory properties to the dictionary while simultaneously training a linear classifier. The classifier is applied on the sparse representations to perform the identification task. The training data are vectorized file descriptors built from large datasets of clean and malware samples. Lacking runtime information, these descriptors try to compensate by including most, if not all, of the information that can be extracted from each sample leading to very long descriptors. The dynamics of the cyber-security field in general, and malware design in particular, urges us to design our models such that device specialization and fast adaptations are possible. A watch will see different attacks than a router and new types of malware are released everyday. Thus we tackle large data samples whose patterns are often changing.

When device specifications are extremely restricting and we deal with large-scale problems, direct DL methods can be prohibitive and instead online approaches can be applied. In online methods, the sparse representation is learned by adapting the dictionary to incoming batches of signals. With each such set, the dictionary is updated as to encompass the new information brought by the signals, without it being recomputed entirely at all times, thus reducing resource demands. Formulated correctly, this continuous dictionary update can be turned to our advantage in adapting to new malware.

With this in mind, we study the case where we start with an initial dictionary trained on a small labeled dataset and then proceed to the online stage where we receive unlabeled data items that we classify but also use to update the dictionary. The initial stage represents supervised learning that can be performed offline, with the resulting model installed on the IoT device. The online stage represents unsupervised learning and is meant to employ fast and incremental dictionary updates such that they can be performed directly on the IoT device. The entire process is called semi-supervised learning. In this paper, our focus is on the unsupervised learning stage.

While the subject of semi-supervised DL has been studied in offline [6] and online [7] contexts, to our knowledge, the problem of updating both the classifier and the dictionary for each incoming unlabeled data item that model receives, has not been studied before. Existing methods use various heuristics, such as classification confidence indicators, to choose only a

This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI-UEFISCDI, project number 17PCCDI/2018 within PNCDI III.

safe subset of the incoming signals for the learning process. This selection discourages new patterns from entering the ML model, which might be desired in some applications, but not for cases such as malware identification where fresh information is key.

II. Preliminaries

A. Classification

The task of instilling class discriminative properties to the dictionary has been extensively studied in the past. We focus here on algorithms that explicitly train a linear classifier and choose Label Consistent K-SVD (LC-KSVD) [8] in particular due to its extensive use and reported success in the literature.

Let the $\boldsymbol{Y}$ training signals belong to any of c known classes, LC-KSVD extends the general DL problem (1) by assigning atom subsets for each class and forcing the representations to follow the class label rather than approximation quality. The new objective becomes

$$\min_{\boldsymbol{D},\boldsymbol{W},\boldsymbol{A},\boldsymbol{X}} \|\boldsymbol{Y}-\boldsymbol{D}\boldsymbol{X}\|_F^2+\alpha\|\boldsymbol{H}-\boldsymbol{W}\boldsymbol{X}\|_F^2+\beta\|\boldsymbol{Q}-\boldsymbol{A}\boldsymbol{X}\|_F^2 \quad (2)$$

where $\boldsymbol{H} \in \mathbb{R}^{c\times N}$ represents the label matrix and $\boldsymbol{Q} \in \mathbb{R}^{n\times N}$ represents the matrix allocating each atom to a specific class. If signal $\boldsymbol{y}$ belongs to class c_i, then column $\boldsymbol{h}_i$ will have element c_i set and zeros everywhere else. The rows in $\boldsymbol{Q}$ correspond to each atom in $\boldsymbol{D}$ and the columns to each signal in $\boldsymbol{Y}$. Let $\boldsymbol{Q}$ be the zero matrix, for each class c_i we fetch the block from $\boldsymbol{Q}$ whose rows correspond to the group of atoms allocated to c_i and whose columns correspond to the batch of signals belonging to class c_i, and set all its entries to 1. Putting aside their special structure, we can interpret $\boldsymbol{H}$ and $\boldsymbol{Q}$ as training signals for the dictionaries $\boldsymbol{W}$ and, respectively, $\boldsymbol{A}$. As $\boldsymbol{D}$ is the sparse representation dictionary producing $\boldsymbol{x}$ given signal $\boldsymbol{y}$, $\boldsymbol{W} \in \mathbb{R}^{c\times n}$ is the class label representation dictionary and is thus a linear classifier for $\boldsymbol{x}$. Dictionary $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is also a linear transformation on the representation $\boldsymbol{x}$ that enforces label consistency instead of approximation quality.

The classification problem (2) can be rewritten through basic algebraic operations in the standard DL form (1) by extending the training signals and dictionary matrices:

$$\min_{\boldsymbol{D},\boldsymbol{W},\boldsymbol{A},\boldsymbol{X}} \left\| \begin{bmatrix} \boldsymbol{Y} \\ \sqrt{\alpha}\boldsymbol{H} \\ \sqrt{\beta}\boldsymbol{Q} \end{bmatrix} - \begin{bmatrix} \boldsymbol{D} \\ \sqrt{\alpha}\boldsymbol{W} \\ \sqrt{\beta}\boldsymbol{A} \end{bmatrix} \boldsymbol{X} \right\|_F^2 \quad (3)$$

This can now be solved through standard DL algorithms, such as K-SVD, where we denote the extended signals with $\tilde{\boldsymbol{Y}}$ and the extended dictionary with $\tilde{\boldsymbol{D}}$. The matrices $\boldsymbol{D}$ and $\boldsymbol{W}$ are extracted at the end of the learning process from $\tilde{\boldsymbol{D}}$ and can now be used for classification; the unlabled signal $\boldsymbol{y}$ is sparsely represented with $\boldsymbol{D}$, by means of an algorithm such as OMP, and then the largest entry of the $\boldsymbol{W}\boldsymbol{x}$ vector decides its class.

B. Online Learning

In the online setting an incoming signal $\boldsymbol{y}$ arrives at time t and we use it to update the current dictionary $\boldsymbol{D}^{(t)}$. The dominant online DL algorithms are ODL [9] and RLS-DLA [10]. ODL is a coordinate descent algorithm that iteratively updates the dictionary atoms in K-SVD fashion; in order to obtain the optimal dictionary multiple sweeps of coordinates need to be performed. RLS-DLA, on the other hand, follows the recursive least square paradigm which ensures optimality with a few simple vector multiplications. In this article we will focus on the later due to its low complexity and encouraging results (with numerical stability modifications) when compared to ODL [2, Chapter 5].

MOD [5] keeps the signals and representations fixed in (1), and updates the dictionary by solving the least squares (LS) problem $\boldsymbol{D}\boldsymbol{X} = \boldsymbol{Y}$. RLS-DLA rephrases this approach in the online context starting from the matrices

$$\boldsymbol{G} = \boldsymbol{X}\boldsymbol{X}^T, \quad \boldsymbol{P} = \boldsymbol{Y}\boldsymbol{X}^T, \quad (4)$$

such that the LS at time t is written as $\boldsymbol{G}\boldsymbol{D}^{(t)} = \boldsymbol{P}$. Given a new signal $\boldsymbol{y}$ at time $t+1$, updating the dictionary resumes to solving

$$\boldsymbol{D}^{(t+1)}(\boldsymbol{G} + \boldsymbol{x}\boldsymbol{x}^T) = \boldsymbol{P} + \boldsymbol{y}\boldsymbol{x}^T. \quad (5)$$

where $\boldsymbol{x}$ is the sparse representation using $\boldsymbol{D}^{(t)}$. This leads to the simple update rule

$$\boldsymbol{D} \leftarrow \boldsymbol{D} + \alpha\boldsymbol{r}\boldsymbol{u}^T \quad (6)$$

where $\boldsymbol{u} = \varphi^{-1}\boldsymbol{G}^{-1}\boldsymbol{x}$, $\alpha = \frac{1}{1+\boldsymbol{x}^T\boldsymbol{u}}$, $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}$ and φ is a forgetting factor. The inversion is efficiently solved through the matrix inversion lemma and the entire update requires at most $O(n^2)$ operations.

C. Online Classification

The LC-RLSDLA [11] algorithm adds the class label and label consistency constraints of LC-KSVD to the RLS-DLA algorithm. When performing (6) in the supervised case, we simply replace the incoming signal and existing dictionary with their extended version, $\tilde{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y}; \sqrt{\alpha}\boldsymbol{h}; \sqrt{\beta}\boldsymbol{q} \end{bmatrix}$ and $\tilde{\boldsymbol{D}} = \begin{bmatrix} \boldsymbol{D}; \sqrt{\alpha}\boldsymbol{W}; \sqrt{\beta}\boldsymbol{A} \end{bmatrix}$, from (3). Thus $\boldsymbol{D}$, $\boldsymbol{W}$ and $\boldsymbol{A}$ are simultaneously updated.

In the unsupervised case, when unlabeled signal $\boldsymbol{y}$ arrives, we do not have access to its label $\boldsymbol{h}$. Let $\boldsymbol{x}$ be the sparse representation of $\boldsymbol{y}$ and let $\tilde{\boldsymbol{h}} = \boldsymbol{W}\boldsymbol{x}$ be the classification result. LC-RLSDLA denotes $c_r = \max_k(\tilde{\boldsymbol{h}}, 1)$ the credibility level and $c_o = (c_r - \max_k(\tilde{\boldsymbol{h}}, 2))/c_r$ the confidence level, where $\max_k(\boldsymbol{v}, i)$ provides the i-th largest entry of vector $\boldsymbol{v}$. If both levels are above certain fixed and user provided thresholds, then vectors $\boldsymbol{h}$ and $\boldsymbol{q}$ are built according to the classification result (as described around (2)) and dictionary $\tilde{\boldsymbol{D}}$ is updated as in the supervised case. If either of the two levels are bellow the threshold, the signal is discarded.

Online Semi-Supervised Dictionary Learning (OSSDL) [7] also adapts LC-KSVD to the online scenario, but uses ODL instead of RLS-DLA: dictionary update is performed through multiple coordinate descent iterations. The extended signal $\tilde{\boldsymbol{y}}$ and dictionary $\tilde{\boldsymbol{D}}$ are identical to the ones used by LC-RLSDLA. When running supervised, OSSDL updates the extended dictionary $\tilde{\boldsymbol{D}}$, but is more cautious when running unsupervised and resumes to only updating the sparse representation dictionary $\boldsymbol{D}$.

Similar to LC-RLSDLA, OSSDL allows only some of the incoming unlabeled signals to enter the learning process based on a classification confidence level. Let c_i be the class assigned to unlabeled signal $\boldsymbol{y}$ through the classification process and let $\boldsymbol{x}$ be its sparse representation. Entry x_j from $\boldsymbol{x}$ is the coefficient of the atom $\boldsymbol{d}_j$ in $\boldsymbol{D}$ and $\boldsymbol{d}_j$ is constrained, through $\boldsymbol{Q}$, to represent signals from a single class. The confidence level is computed by looking at the magnitude of the entries in $\boldsymbol{x}$ corresponding to class c_i in relation to the rest of the entries. Take two fixed user provided thresholds ϕ_{low} and ϕ_{high} corresponding to low and high confidence levels. If the confidence level is above ϕ_{high} $\boldsymbol{D}$, $\boldsymbol{W}$ and $\boldsymbol{A}$ are updated as if running supervised. If the current confidence level is between the two thresholds, only $\boldsymbol{D}$ is updated using standard ODL. If the confidence level is beneath ϕ_{low} the user is prompted to provide a label.

In [7], OSSDL performs DL on both labeled and unlabeled data. At the end of the learning process, classifier $\boldsymbol{W}$ is updated separately using only labeled data following the solution of the regularized LS problem $\|\boldsymbol{H} - \boldsymbol{W}\boldsymbol{X}\|_F^2 + \lambda \|\boldsymbol{W}\|_F^2$. The updated classifier is then used to reclassify the unlabeled data.

This process would not be feasible in our case because we only receive unlabeled data online and thus find the task of storing and recomputing the classification of the entire history intractable and perhaps even futile. If at time t we label a sample clean and, through reclassification, we consider it malware at time $t' > t$, there is not much we can do about our initial verdict and the actions performed by the IoT device as consequence to our judgement at time t. None the less, in our experiments we modify OSSDL to fit our scenario, even though it was not designed as such, which should be taken into consideration when studying its numerical results.

III. Our Method

Starting from a pre-trained dictionary state, our goal is to provide a fast, tolerant and unsupervised online method. Fast because we are running online on low-powered IoT devices, tolerant as we want to permit all incoming signals to update the dictionary, and unsupervised because we have no label information and do not rely on user intervention in any way.

These properties are key for our malware identification problem. We want a fast algorithm that can perform the identification task directly on the device, so it requires low computational complexity. Hence we prefer RLS-DLA over ODL. We want to continuously update our DL model so that our classifier can learn and adapt to new threats, thus we have to be tolerant to all incoming signals, especially to the ones with low classification confidence levels, because they provide fresh information. Threshold selection strategies, such as the ones used by SSDL and LC-RLSDLA, deprive us of fresh information. Finally, we want to run unsupervised due to privacy and usability concerns, we want to be self-sufficient, never externalizing data to third party entities and never asking the IoT end-user to decide for us. This rules out manually labeling samples with low classification confidence as in the SSDL case.

We base our method on LC-RLSDLA, where the credibility and confidence thresholds are set to zero. Permitting all unlabeled signals to enter the dictionary update stage is a challenging task and we employ multiple strategies to make this action feasible.

No matter how malware and clean samples are described, they are fundamentally identical up to a point because they are both executables that have to respect file format constraints such as program headers, data sections and library imports. To account for this commonality and focus only on the distinctiveness of the clean and malware classes, we modify the label consistency constraint to permit a set of dictionary atoms to partake in the representation of both classes. Having atoms $\boldsymbol{d}_1$ represent the clean class, $\boldsymbol{d}_2$ represent the malware class, and $\boldsymbol{d}_3$ to represent their shared features, the label consistent matrix $\boldsymbol{Q}$ can be resumed to two columns $\boldsymbol{q}_1 = [1\ 0\ 1]^T$ and $\boldsymbol{q}_2 = [0\ 1\ 1]^T$ corresponding to clean and malware samples respectively. When running online, we need only store a column for each class $\boldsymbol{Q} \in \mathbb{R}^{n \times c}$.

In order to dampen the effect that incoming signal $\boldsymbol{y}$ has on the current model, we extend the objective in (2) and rephrase it for the online setting by regularizing the change in $\boldsymbol{W}$ and $\boldsymbol{A}$ (thus also implicitly in $\boldsymbol{D}$):

$$\begin{aligned}\min_{\boldsymbol{D},\boldsymbol{W},\boldsymbol{A}} & \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}\|_2^2 + \alpha\|\boldsymbol{h} - \boldsymbol{W}\boldsymbol{x}\|_2^2 + \beta\|\boldsymbol{q} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \\ & + \lambda_1 \|\boldsymbol{W} - \boldsymbol{W}_0\|_F^2 + \lambda_2 \|\boldsymbol{A} - \boldsymbol{A}_0\|_F^2\end{aligned} \tag{7}$$

Here $\boldsymbol{h}$ is the unit vector $\boldsymbol{e}_i$, where position i is decided based on the classification result, $\boldsymbol{q}$ is the i-th column from $\boldsymbol{Q}$, Matrices $\boldsymbol{W}_0$ and $\boldsymbol{A}_0$ represent the current classifier and label consistency dictionary.

Because the objective in (7) does not have a direct solution, we first update the dictionaries without regularization ($\lambda_1 = \lambda_2 = 0$) which leads to the plain LC-RLSDLA problem. The resulting dictionaries are $\boldsymbol{D}_0$, $\boldsymbol{W}_0$ and $\boldsymbol{A}_0$. We proceed by mediating the rate of change in $\boldsymbol{W}_0$ and $\boldsymbol{A}_0$ through regularization. We define

$$f(\boldsymbol{W}) = \|\boldsymbol{h} - \boldsymbol{W}\boldsymbol{x}\|_2^2 + \lambda_1 \|\boldsymbol{W} - \boldsymbol{W}_0\|_F^2 \tag{8}$$

$$g(\boldsymbol{A}) = \|\boldsymbol{q} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda_2 \|\boldsymbol{A} - \boldsymbol{A}_0\|_F^2 \tag{9}$$

as the minimization objective for $\boldsymbol{W}$ and $\boldsymbol{A}$ respectively, when considering everything else fixed in (7). Here we make an abuse of notation and set $\lambda_1 = \lambda_1/\alpha$ and $\lambda_2 = \lambda_2/\beta$. We can view f and g as proximal functions [12] where $\boldsymbol{W}_0$ and $\boldsymbol{A}_0$ are obtained from the previous iteration. The solutions to (8) and (9) are the simple LS problems

$$\boldsymbol{W} = (\boldsymbol{h}\boldsymbol{x}^T + \lambda_1\boldsymbol{W}_0)(\boldsymbol{x}\boldsymbol{x}^T + \lambda_1\boldsymbol{I})^{-1} \tag{10}$$

$$\boldsymbol{A} = (\boldsymbol{q}\boldsymbol{x}^T + \lambda_2\boldsymbol{A}_0)(\boldsymbol{x}\boldsymbol{x}^T + \lambda_2\boldsymbol{I})^{-1} \tag{11}$$

obtained from the gradient equations $\nabla f = 0$ and $\nabla g = 0$.

If we look at f and g as generalized Tikhonov regularizations, the Tikhonov parameters λ_1 and λ_2 can be estimated through generalized cross-validation [13]. In our case, we have to perform the estimation for each new signal which is prohibitive due to its high computational cost. Instead we propose two alternatives. Either, we fix λ_1 and λ_2 in the supervised phase through standard cross-validation, this is the fastest option that lifts the estimation burden when running online and unsupervised. Or, given that the Tikhonov parameter is thightly related to the singular values of $\boldsymbol{x}\boldsymbol{x}^T$, we look at the change brought to (4) in (5) as a rank-1 update to positive semidefinite matrix $\boldsymbol{G}$. The effect of the update is reflected in the spectrum of $\varphi\boldsymbol{G} + \boldsymbol{x}\boldsymbol{x}^T$, which indirectly influences that of $\boldsymbol{W}_0$ and $\boldsymbol{A}_0$, and which can be iteratively updated when a new signal arrives [14]. We have seen good results in our experiments when setting

$$\lambda_{1,2} = \|\boldsymbol{G}\|_2 \quad \text{or} \quad \lambda_1 = \|\boldsymbol{W}_0\|_2\,,\ \lambda_2 = \|\boldsymbol{A}_0\|_2\,. \tag{12}$$

Algorithm 1: TODDLeR

```
Data: signal $\boldsymbol{y} \in \mathbb{R}^m$, sparsity level $s \in \mathbb{R}$
      dictionaries $\boldsymbol{D} \in \mathbb{R}^{m\times n}$, $\boldsymbol{W} \in \mathbb{R}^{c\times n}$, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$
      label consistency matrix $\boldsymbol{Q} \in \mathbb{R}^{n\times c}$
      parameters $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$
Result: updated dictionaries $\boldsymbol{D}$,$\boldsymbol{W}$,$\boldsymbol{A}$
1 Representation: $\boldsymbol{x} = \text{OMP}(\boldsymbol{y}, \boldsymbol{D}, s)$
2 Classification: $i = \arg\max_{j=1:c}(\boldsymbol{W}\boldsymbol{x})$
3 Build labels: $\boldsymbol{h} = \boldsymbol{e}_i$, $\boldsymbol{q} = \boldsymbol{q}_i$ s.t. $\tilde{\boldsymbol{y}} = [\ \boldsymbol{y}; \sqrt{\alpha}\boldsymbol{h}; \sqrt{\beta}\boldsymbol{q}\ ]$
4 Dictionary update: $\tilde{\boldsymbol{D}} = \text{RLS-DLA}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{D}}, s)$
5 Optional tuning: set $\lambda_{1,2}$ according to (12)
6 Regularization: set $\boldsymbol{W}$ and $\boldsymbol{A}$ according to (11)
```

The operations performed at the arrival of a new unlabeled signal $\boldsymbol{y}$ are described in Algorithm 1 called Tolerant Online Discriminative DL with Regularization (TODDLeR). Step 1 computes the sparse representation $\boldsymbol{x}$ using OMP needed for applying the linear classifier in step 2: signal $\boldsymbol{y}$ belongs to class i corresponding to the position of the maximum value in $\boldsymbol{W}\boldsymbol{x}$. Step 3 produces the extended signal $\tilde{\boldsymbol{y}}$, that includes the label and label consistent vectors of class i, to be used when updating the extended dictionary $\tilde{\boldsymbol{D}} = [\boldsymbol{D}; \boldsymbol{A}; \boldsymbol{W}]$ in step 4. We use RLS-DLA to perform online DL. Steps 1–4 can be viewed as performing LC-RLSDLA with nil thresholds. Unless fixed beforehand, step 5 updates the regularization parameters as earlier described. Step 6 performs the actual regularization on $\boldsymbol{W}$ and $\boldsymbol{A}$. Counting the computational intensive operations we have $O(smn)$ instructions for OMP in step 1, $O(n^2)$ for RLS-DLA in step 4, and $O(s^3)$ for efficiently solving the two LS problems in step 6. Tuning the regularization parameters adds an extra cost of $O(n^2)$ operations required for computing the new eigenvalues in (12). This leads to a total of $O(smn + n^2 + s^3)$ operations.

IV. Results

We have chosen two existing datasets for testing TODDLeR. A newer, more complex dataset of Windows Portable Executable (PE) files and an older, highly popular Android malware database. The two differ considerably in dimension, class balance and complexity of feature space. Nonetheless, whenever possible, we kept the same algorithm parameters across the two cases, with an eye on reliability rather than an ideal fine-tuning to dataset specifics.

Offline pre-training was performed in similar conditions, on a small signal batch. We used LC-KSVD to train a dictionary of $n = 3m$ atoms, where an equal number of m atoms were allocated for the clean and malware classes, and for the shared sub-dictionary. At initialization, 20 DL iterations are separately performed on these 3 atom groups, followed by 50 iterations of training on the entire dictionary.

We used $\alpha = 4$ and $\beta = 16$ [8] for both LC-KSVD and LC-RLSDLA. This choice is also confirmed by a preliminary cross-validation we performed on the first dataset, where $\alpha = 16$, $\beta = 16$ resulted in similar performance. For RLS-DLA, we set the forgetting factor to $\varphi = 0.999$. The credibility and confidence thresholds in LC-RLSDLA were fixed to $c_o = 0.2$, $c_r = 0.7$ [11] and the high and low thresholds in OSSDL were set to $\phi_{low} = 3$ and $\phi_{high} = 4.5$ [7].

A. Execution time

We compared execution times on synthetic data $\boldsymbol{Y} \in \mathbb{R}^{50\times 1000}$ with sparsity and dictionary overcompleteness as described above. We report average values over 100 trials. All algorithms were implemented in Matlab R2018b and performed on an Intel i5 CPU with 8GB of system memory.

The lowest execution time was obtained with LC-RLSDLA, 2.327ms per signal, followed by TODDLeR without parameter tuning at 3.381ms. The two parameter tuning strategies, $\lambda_{1,2} = \|\boldsymbol{G}\|_2$ and $\lambda_1 = \|\boldsymbol{W}\|_2$ $\lambda_2 = \|\boldsymbol{A}\|_2$ were executed in 6.251ms and 6.904ms respectively, while OSSDL took 11.572ms.

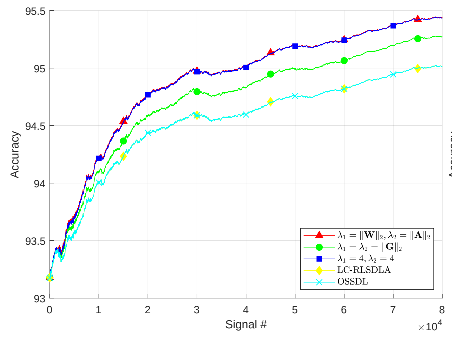
B. DREBIN Dataset

The DREBIN dataset [15] consists of 123,453 legitimate Android applications and 5,560 malware. The authors performed static analysis on the files, that resulted in $m = 11$ separate features stemming from 8 categories that span application hardware requests, permission requests, inter- and intra-process communications, Android restricted API calls, network connections and others.

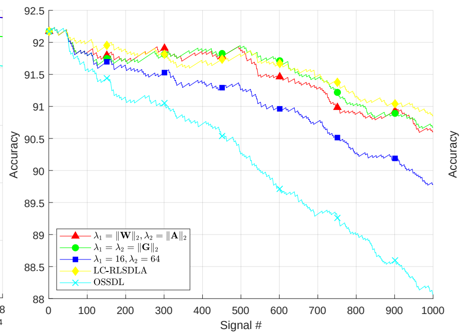
The training and test signals were obtained by performing a hashing vectorization, with 11 bins, using the Python scikit-learn feature extraction module. Unlike [15], we work with the 11 individual features directly, rather than the 8 category features.

We reserved 80,000 signals for testing and used the remainder for pre-training. LC-KSVD training was performed on 40,000 signals, with $s = 5$, and resulted in 95.87% accuracy when tested on a set of 9,013 samples. Each of these splits ensured a fair amount of clean and malware samples.

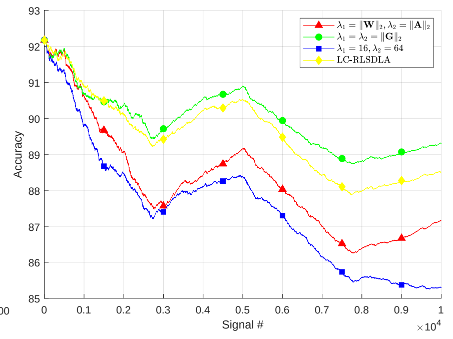
Online training with TODDLeR was performed in different regularization setups. First, fixed values for λ_1 and λ_2 were considered, as they provide a fast solution, despite requiring preliminary parameter adjustment. Parameters $\lambda_{1,2} = 4$ yielded best results and are reported here. Parameter tuning with $\lambda_1 = \|\boldsymbol{W}\|_2$ $\lambda_2 = \|\boldsymbol{A}\|_2$ yielded similar accuracy,



(a) Classification accuracy on Drebin dataset



(b) Preliminary test on Ember dataset



(c) Classification accuracy on Ember dataset

while $\lambda_{1,2} = \|\boldsymbol{G}\|_2$ came in second. Either way, TODDLeR outperformed both LC-RLSDLA and OSDDL as presented in Figure 1a.

C. EMBER Dataset

The EMBER dataset [16] was created as a static malware detection benchmarking tool for machine learning methods, in an attempt to overcome the lack of standardization and maintained datasets in the field. It contains 1.1 million features of clean, malware and unlabeled PE files. The feature space is $m = 2351$ and includes header information, imported functions and 256 features representing the byte histogram of the file. In our experiments we use clean and malware data.

First we perform z-score sample standardization to ensure a democratic treatment of all features. LC-KSVD pre-training, with $s = 20$, was executed on 7,200 signals and resulted in 92.17% accuracy when tested on a set of 1,800 samples. The resulting dictionary was used for TODDLeR initialization.

We test three different regularization strategies on 10,000 test signals and compare the results with LC-RLSDLA and OSSDL. Figure 1b shows the classification accuracy after 1,000 signals. We can see that LC-RLSDLA behaves similarly to TODDLeR, while OSSDL falls behind. We continue the experiment with LC-RLSDLA. Figure 1c shows the classification accuracy results over the entire set. LC-RLSDLA comes in second after TODDLeR with parameter tuning $\lambda_{1,2} = \|\boldsymbol{G}\|$, while tuning based on matrices $\boldsymbol{W}$ and $\boldsymbol{A}$ and fixed parametrization come in last. The ups and downs are due to new malware types that temporarily hinder performance.

Area Under the Curve (AUC) values confirm the hierarchy in the figures above. The highest value, 0.8878, is obtained when regularization is performed with $\lambda_{1,2} = \|\boldsymbol{G}\|_2$, followed by LC-RLSDLA with 0.8782. Third is parameter update with $\lambda_1 = \|\boldsymbol{W}\|_2$ $\lambda_2 = \|\boldsymbol{A}\|_2$, with 0.8626, while the fixed $\lambda_{1,2}$ strategy has an AUC score of 0.8406.

V. Conclusions

We proposed a semi-supervised framework in which an initial dictionary is trained on labeled data using standard DL algorithms and developed a new algorithm for the second stage where the dictionary is continuously updated in online fashion with new unlabled signals. The elements of novelty are the regularization constraint during online training, the fact that, unlike existing methods, we use all unlabeled signals to update the dictionary, and the application of DL to malware identification with encouraging numerical results.

References

[1] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, “IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?” *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, Sept 2018.

[2] B. Dumitrescu and P. Irofti, *Dictionary Learning Algorithms and Applications*. Springer, 2018.

[3] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, “Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition,” in *27th Asilomar Conf. Signals Systems Computers*, vol. 1, Nov. 1993, pp. 40–44.

[4] M. Aharon, M. Elad, and A. Bruckstein, “K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation,” *IEEE Trans. Signal Proc.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[5] K. Engan, S. Aase, and J. Husoy, “Method of optimal directions for frame design,” in *IEEE Int. Conf. Acoustics Speech Signal Proc.*, vol. 5, 1999, pp. 2443–2446.

[6] X. Liu, M. Song, D. Tao, X. Zhou, C. Chen, and J. Bu, “Semi-supervised coupled dictionary learning for person re-identification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3550–3557.

[7] G. Zhang, Z. Jiang, and L. Davis, “Online semi-supervised discriminative dictionary learning for sparse representation,” in *Asian conference on computer vision*. Springer, 2012, pp. 259–273.

[8] Z. Jiang, Z. Lin, and L. Davis, “Learning A Discriminative Dictionary for Sparse Coding via Label Consistent K-SVD,” in *IEEE Conf. Computer Vision and Pattern Recognition*, 2011, pp. 1697–1704.

[9] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, “Online Learning for Matrix Factorization and Sparse Coding,” *J. Machine Learning Res.*, vol. 11, pp. 19–60, Jan. 2010.

[10] K. Skretting and K. Engan, “Recursive least squares dictionary learning,” *IEEE Trans. Signal Proc.*, vol. 58, no. 4, pp. 2121–2130, 2010.

[11] S. Matiz and K. Barner, “Label consistent recursive least squares dictionary learning for image classification,” in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 1888–1892.

[12] G. Peng and W. Hwang, “A proximal method for dictionary updating in sparse representations,” *IEEE Transactions on Signal Processing*, vol. 63, no. 15, pp. 3946–3958, 2015.

[13] G. H. Golub, M. Heath, and G. Wahba, “Generalized cross-validation as a method for choosing a good ridge parameter,” *Technometrics*, vol. 21, no. 2, pp. 215–223, 1979.

[14] J. R. Bunch, C. P. Nielsen, and D. C. Sorensen, “Rank-one modification of the symmetric eigenproblem,” *Numerische Mathematik*, vol. 31, no. 1, pp. 31–48, 1978.

[15] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of android malware in your pocket,” 2014.

[16] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” *ArXiv e-prints*, 2018.