DL4MD: A Deep Learning Framework for Intelligent Malware Detection

William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye*, and Xin Li
Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506, USA

Abstract—In the Internet-age, malware poses a serious and evolving threat to security, making the detection of malware of utmost concern. Many research efforts have been conducted on intelligent malware detection by applying data mining and machine learning techniques. Though great results have been obtained with these methods, most of them are built on shallow learning architectures, which are still somewhat unsatisfying for malware detection problems. In this paper, based on the Windows Application Programming Interface (API) calls extracted from the Portable Executable (PE) files, we study how a deep learning architecture using the stacked AutoEncoders (SAEs) model can be designed for intelligent malware detection. The SAEs model performs as a greedy layerwise training operation for unsupervised feature learning, followed by supervised parameter fine-tuning (e.g., weights and offset vectors). To the best of our knowledge, this is the first work that deep learning using the SAEs model based on Windows API calls is investigated in malware detection for real industrial application. A comprehensive experimental study on a real and large sample collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. Promising experimental results demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning methods.

Keywords: Malware Detection, Windows Application Programming Interface (API) Calls, Deep Learning.

1 Introduction

Malware is malicious software disseminated to infiltrate the secrecy, integrity, and functionality of a system [8], such as viruses, worms, trojans, backdoors, spyware. With computers and the Internet being essential in everyday life, malware poses a serious threat to their security. It is estimated that one in four computers operating in the U.S. are infected with malware [24]. The threat malware poses is not only emotional, but financial as well. According to a recent report from Kaspersky Lab, up to one billion dollars was stolen in roughly two years from financial institutions worldwide, due to malware attacks [16]. As a result, the detection of malware is of major concern to both the anti-malware industry and researchers.

In order to protect legitimate users from the attacks, the majority of anti-malware software products (e.g., Comodo, Symantec, Kaspersky) use the signature-based method of detection [10], [9]. Signature is a short string of bytes, which is unique for each known malware so that its future examples can be correctly classified with a small error rate [19]. However, this method can be easily evaded by malware attackers through the techniques such as encryption, polymorphism and obfuscation [29], [2]. Furthermore, malicious files are being disseminated at a rate of thousands per day [34], making it difficult for this signature-based method to be effective. In order to combat the malware attacks, intelligent malware detection techniques need to be investigated. As a result, many researches have been conducted on intelligent malware detection by applying data mining and machine learning techniques in recent years [27], [1], [23], [32], [35]. Based on different feature representations, different kinds of classification methods, such as Artificial Neural Network (ANNs), Support Vector Machine (SVM), Naïve Bayes (NB), and Decision Tree (DT), are used for model construction to detect malware [20], [7], [27]. Most of these methods are built on shallow learning architectures. Though they had isolated success in malware detection, shallow learning architectures are still somewhat unsatisfying for malware detection problems.

Deep learning (DL), a new frontier in data mining and machine learning, is starting to be leveraged in industrial and academic research for different applications (e.g., Computer Vision) [4], [14], [22]. A multilayer deep learning architecture is of superior ability in feature learning. More importantly, deep learning architectures overcome the learning difficulty through layerwise pretraining, i.e. pretraining multiple layers of feature detectors from the lowest level to the highest level to construct the final classification model [22]. This inspires us to devise a malware detection architecture based on deep learning.

In this paper, a deep learning architecture using the stacked AutoEncoders (SAEs) model for malware detection is studied, with the input resting on Windows Application Programming Interface (API) calls extracted from the Portable Executable (PE) files. The SAEs model employs a greedy layerwise training operation for unsupervised feature learning, followed by supervised parameter fine-tuning (e.g., weights and offset vectors). To the best of our knowledge, this is the first work investigating deep learning using the SAEs model resting on Windows API calls in malware detection for real industrial application. For validation, a comprehensive experimental study on a real and large sample collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. The experimental results obtained demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning methods.
methods and that deep learning is a promising architecture for malware detection.

The rest of the paper is structured as follows. Section 2 presents the related work. Section 3 describes the overview of our malware detection system. Section 4 introduces our proposed deep learning approach for malware detection. Section 5 evaluates the performance of our proposed method in comparison with other alternative methods in malware detection. Finally, Section 6 concludes.

2 Related work

Signature-based method is widely used in anti-malware industry [10], [9]. However, malware authors can easily evade this signature-based method through techniques such as encryption, polymorphism and obfuscation [29]. Driven by the economic benefits, the quantity, diversity and sophistication of malware has significantly increased in recent years [34]. In order to effectively and automatically detect malware from the real and large daily sample collection, new, intelligent malware detection systems have been developed by applying data mining and machine learning techniques [31], [27], [20], [33], [28], [13]. These intelligent malware detection systems are varied in their uses of feature representations and classification methods. Naïve Bayes on the extracted strings and byte sequences was applied in [27], which claimed that Naïve Bayes classifier performed better than traditional signature-based method. Kolter et al. [20] focused on static analysis of the executable files and compared Naïve Bayes, Support Vector Machine and Decision Tree based on the \( n \)-grams. Ye et al. [33] proposed IMDS performing associative classification on Windows API calls extracted from executable files. Shah et al. [28] applied various feature selection algorithms to obtain the feature sets from PE files and used Artificial Neural Networks to detect new and unknown malware. Hou et al. [13] developed the intelligent malware detection system using cluster-oriented ensemble classifiers resting on the analysis of Windows API calls. Most of these existing researches are built on shallow learning architectures.

Due to its superior ability in feature learning through multilayer deep architecture [11], deep learning is feasible to learn higher level concepts based on the local feature representations [25]. As a result, researchers have paid much attention to deep learning methods in the domains of natural language processing, computer vision, etc. [18], [6]. In recent years, limited research efforts have been devoted to the malware detection using deep learning [21], [25], [15]. Ouellette et al. [25] extracted control flow graphs to present malware samples, and used a deep probabilistic model (sum-product network) to compare the similarities between the unknown file samples and those of representative sample features from known classes of malware. Jung et al. [15] used the features of header, tags, bytecode and API calls and utilized an ensemble learner consisting of different deep learning networks (e.g., deep feed-forward neural network, deep recurrent neural network) to classify the Adobe Flash file samples. Li et al. [21] proposed a hybrid malicious code detection approach on the basis of AutoEncoder and Deep Belief Network, where AutoEncoder was used to reduce the dimensionality of data, and a Deep Belief Network was applied to detect malicious code.

Different from the previous work, based on a real and large sample collection from Comodo Cloud Security Center, resting on the analysis of Windows API calls, we attempt to explore a deep learning architecture with the SAEs model to learn generic features of malware and thus to detect newly unknown malware.

3 System architecture

Our deep learning framework for malware detection (short for DL4MD) is performed on the analysis of Windows API calls generated from the collected PE files. The system consists of two major components: feature extractor, and deep learning based classifier, as illustrated in Figure 1.

![Fig. 1. DL4MD system architecture](image)

- **Feature Extractor**: It is composed of Decompressor and PE Parser. The Decompressor needs to be employed first when a PE file is previously compressed by a binary compress tool (e.g., UPX and ASPack Shell) or embedded a homemade packer. The PE parser is used to extract the Windows API calls from each PE file. Through the API query database, the Windows API calls can be converted to a set of 32-bit global IDs representing the corresponding API functions (e.g., the API of “MAPI32.MAPIReadMail” is encoded as \( 0x00600F12 \)). Then the API calls are stored as the signatures of the PE files in the signature database.

- **Deep Learning based Classifier**: Based on the Windows API calls, a deep learning architecture using SAEs model is designed to perform unsupervised feature learning, supervised fine-tuning, and thus malware detection. (See Section 4 for details.)
4 Proposed method

4.1 Problem definition

Relying on the analysis of Windows API calls, which can reflect the behavior of program code pieces [32] (e.g., the API “GetModuleFileNameA” in “Kernel32.DLL” can be used to retrieve the complete path of the file that contains the specified module of current process), a file \( f_i \) in our dataset \( D \) is denoted to be of the form \( \langle A_{f_i}, C_{f_i} \rangle \), where \( A_{f_i} \) is the Windows API call feature vector of file \( f_i \), and \( C_{f_i} \) is the class label of file \( f_i \) (where \( C_{f_i} \in \{ c_m, c_b, c_g \} \); \( c_m \) denotes malicious, \( c_b \) benign, and \( c_g \) unknown). Let \( d \) be the number of all extracted Windows API calls in the dataset \( D \). The feature of each file can be represented by a binary feature vector:

\[
A_{f_i} = \langle a_{i1}, a_{i2}, a_{i3}, \ldots, a_{id} \rangle, \tag{1}
\]

where \( a_{ij} = \{0, 1\} \) (i.e., if \( f_i \) contains \( API_j \), \( a_{ij} = 1 \); otherwise, \( a_{ij} = 0 \)). Table I shows a sample example.

Our malware detection problem statement can be stated as follows: Given a dataset \( D \) as defined above, assign a label (i.e., \( c_m \) or \( c_b \)) to each unlabeled file (i.e., \( c_g \)) based on its feature vector.

Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Naive Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning models and can thus achieve comparable or better performance [11]. Typical deep learning models include stacked AutoEncoders (SAEs), Deep Belief Networks with Restricted Boltzmann Machine, Convolutional Neural Networks [5], etc. In this paper, we explore a deep learning architecture with SAEs model for malware detection. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network.

4.2 AutoEncoder

An AutoEncoder, also called AutoAssociator, is an artificial neural network used for learning efficient codings [30]. Architecturally, the form of an AutoEncoder is with an input layer, an output layer and one or more hidden layers connecting them. The goal of an AutoEncoder is to encode a representation of the input layer into the hidden layer, which is then decoded into the output layer, yielding the same (or as close as possible) value as the input layer [4]. In this way, the hidden layer acts as another representation of the feature space, and in the case when the hidden layer is narrower (has fewer nodes) than the input/output layers, the activations of the final hidden layer can be regarded as a compressed representation of the input [4], [30], [5]. Figure 2 illustrates a one-layer AutoEncoder model with one input layer, one hidden layer, and one output layer.

![AutoEncoder Diagram](image)

FIG. 2. A one-layer AutoEncoder model

Given a training set \( X \) of \( n \) samples \( \{x_1, x_2, x_3, \ldots, x_n\} \), where \( x_i \) \((i \in \{1, \ldots, n\}) \) is a \( d_0 \)-dimensional feature vector, i.e., \( x_i \in \mathbb{R}^{d_0} \), and a sigmoid function denoted as

\[
s(t) = \frac{1}{1 + \exp^{-t}}, \tag{2}
\]

the AutoEncoder framework can be rigorously defined as follows [30], [22].

**Encoder:** In order to transform an input vector \( x_i \) into a hidden representation vector \( y_i \), the encoder, a deterministic mapping \( f_o \), is utilized. Its typical form is an affine mapping followed by a non-linearity [30]:

\[
y_i = f_o(x_i) = s(Wx_i + b), \tag{3}
\]

where \( W \) is a \( d_0 \times d_1 \) weight matrix, \( d_1 \) is the number of hidden units, \( b \) is an offset vector of dimensionality \( d_1 \), and \( \theta \) is the mapping parameter set \( \theta = \{W, b\} \).

**Decoder:** The resulting hidden representation \( y_i \) is then mapped back to a reconstructed \( d_0 \)-dimensional vector \( z_i \) in the input space, using the decoder \( g_w \). Its typical form is again an affine mapping optionally followed by a non-linearity [30], i.e., either \( z_i = g_w(y_i) = W'y_i + b' \) or

\[
z_i = g_w(x_i) = s(W'y_i + b'), \tag{4}
\]

where \( W' \) is a \( d_1 \times d_0 \) weight matrix, \( b' \) is also an offset vector of dimensionality \( d_0 \), and \( \theta' = \{W', b'\} \).

Typically, the number of hidden units is much less than the number of visible (input/output) ones (\( d_1 < d_0 \)). As a result, when passing data through such a network, it first compresses (encodes) input vector to “fit” in a smaller representation, and then tries to reconstruct (decode) it back. The task of training is to minimize an error or reconstruction (using Equation 5), i.e. find the most efficient compact representation (encoding) for input data (Equation 6).

\[
E(x, z) = \frac{1}{2} \sum_{i=1}^{n} ||x_i - z_i||^2, \tag{5}
\]

\[
\theta = \{W, b\} = \arg\min_{\theta} E(x, z). \tag{6}
\]
Algorithm 1 illustrates the training procedure of AutoEncoder in our application. Note that other parameterized functions and loss functions can also be used for the encoder or decoder in the AutoEncoder framework [30].

Input: Data set \( X \) with \( n \) training samples:
\[
\mathbf{x}_i = <\mathbf{A}_f, \mathbf{C}_f>, \quad \text{where } i \in \{1, \ldots, n\} \text{ and } \mathbf{C}_f \in \{c_m, c_b\}
\]

Output: Parameter set \( \theta = \{\mathbf{W}, \mathbf{b}\} \)

Initialize \( (\mathbf{W}, \mathbf{b}) \);
while training error \( E \) hasn’t converged or the designated iteration hasn’t reached do
  for each input \( \mathbf{x}_i \) do
    Compute activations \( \mathbf{y}_i \) at the hidden layer, and obtain an output \( \mathbf{z}_i \) at the output layer;
  end
  Calculate the training error \( E(\mathbf{x}, \mathbf{z}) \);
  Backpropagate the error through the net and update parameter set \( \theta = \{\mathbf{W}, \mathbf{b}\} \);
Algorithm 1: The algorithm for training AutoEncoder in malware detection

4.3 Deep learning architecture with SAEs

To form a deep network, an SAE model is created by daisy chaining AutoEncoders together, known as stacking: the output of one AutoEncoder in the current layer is used as the input of the AutoEncoder in the next [5]. More rigorously, with an SAE deep network with \( h \) layers, the first layer takes the input from the training dataset and is trained simply as an AutoEncoder. Then, after the \( k^{th} \) hidden layer is obtained, its output is used as the input of the \( (k + 1)^{th} \) hidden layer, which is trained similarly. Finally, the \( h^{th} \) layer’s output is used as the output of the entire SAE model. In this manner, AutoEncoders can form a hierarchical stack. Figure 3 illustrates a SAEs model with \( h \) hidden layers.

To use the SAEs for malware detection, a classifier needs to be added on the top layer. In our application, the SAEs and the classifier comprise the entire deep architecture model for malware detection, which is illustrated in Figure 4.

Algorithm 2: The algorithm for training the deep learning network with SAEs in malware detection

5 Experimental results and analysis

In this section, we conduct two sets of experiments based on a real and large sample collection obtained from Comodo Cloud Security Center to fully evaluate the performance of our deep learning model in malware detection: (1) In the first
TABLE II

PERFORMANCE MEASURES IN MALWARE DETECTION

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>Number of files correctly classified as malicious</td>
</tr>
<tr>
<td>TN</td>
<td>Number of files correctly classified as benign</td>
</tr>
<tr>
<td>FP</td>
<td>Number of files mistakenly classified as malicious</td>
</tr>
<tr>
<td>FN</td>
<td>Number of files mistakenly classified as benign</td>
</tr>
<tr>
<td>TP Rate (TPR)</td>
<td>( \frac{TP}{TP+FN} )</td>
</tr>
<tr>
<td>FP Rate (FPR)</td>
<td>( \frac{FP}{FP+TN} )</td>
</tr>
<tr>
<td>Accuracy (ACY)</td>
<td>( \frac{TP+TN}{TP+TN+FP+FN} )</td>
</tr>
</tbody>
</table>


In this set of experiments, we evaluate the deep learning networks with different parameters (i.e., different numbers of hidden layers and neurons); (2) In the second set of experiments, we conduct a comparison between our proposed method and other shallow learning based classification methods (i.e., ANN, SVM, NB, and DT).

5.1 Experimental setup

The dataset obtained from Comodo Cloud Security Center contains 50,000 file samples, where 22,500 are malware, 22,500 are benign files, and 5,000 are unknown (with the analysis by the anti-malware experts of Comodo Security Lab, 2,500 of them are labeled as malware and 2,500 of them are benign). In our experiments, those 45,000 file samples are used for training, while the 5,000 unknown files are used for testing. 9,649 Windows API calls are extracted from these 50,000 file samples, so all the file samples can be represented as binary feature vectors with 9,649-dimensions (described in Section 4.1). To quantitatively validate the experimental results, we use the performance measures shown in Table II. All experiments are conducted in the environment: 64 Bit Windows 8.1 on an Intel (R) Core (TM) i7-4790 Processor (3.60GHz) with 16GB of RAM, using MySQL and C++.

5.2 Evaluations of different deep networks

In this section, based on the dataset described in Section 5.1, we evaluate the deep learning networks with different parameters (i.e., different numbers of hidden layers and neurons). The results in Table III and Figure 5 demonstrate the effectiveness of our proposed algorithm in malware detection. The deep learning model with 3 hidden layers and 100 neurons at each hidden layer is superior to other deep networks in training and testing accuracy for malware detection.

5.3 Comparisons between deep learning and other shallow learning based classification methods

In this section, using the same dataset described in Section 5.1, we conduct a comparison between our proposed deep learning framework (DL4MD) and other shallow learning based classification methods (i.e., Artificial Neural Network (ANN), Support Vector Machine (SVM), Naïve Bayes (NB), and Decision Tree (DT)) in malware detection. The results in Table IV, Figure 6 and Figure 7 show that our proposed deep learning framework (DL4MD) outperform ANN, SVM, NB, and DT in malware detection.

Deep learning framework (DL4MD) also performs well in detection efficiency: it just takes about 0.1 second for each unknown sample prediction, including feature extraction. This makes our system a practical solution for intelligent malware detection in real industrial application.
### TABLE III
**Evaluation of Different Deep Networks**

#### Training

<table>
<thead>
<tr>
<th>Hidden Layers</th>
<th>Neurons</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>ACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[50 50]</td>
<td>21,859</td>
<td>1,434</td>
<td>21,066</td>
<td>641</td>
<td>0.9539</td>
</tr>
<tr>
<td>2</td>
<td>[100 100]</td>
<td>22,142</td>
<td>1,460</td>
<td>21,040</td>
<td>358</td>
<td>0.9596</td>
</tr>
<tr>
<td>3</td>
<td>[50 50 50]</td>
<td>22,110</td>
<td>1,295</td>
<td>21,205</td>
<td>390</td>
<td>0.9626</td>
</tr>
<tr>
<td>3</td>
<td>[100 100 100]</td>
<td>22,035</td>
<td>953</td>
<td>21,547</td>
<td>465</td>
<td>0.9685</td>
</tr>
<tr>
<td>4</td>
<td>[100 100 100 100]</td>
<td>22,150</td>
<td>1,178</td>
<td>21,322</td>
<td>350</td>
<td>0.9660</td>
</tr>
<tr>
<td>5</td>
<td>[100 100 100 100 100]</td>
<td>22,055</td>
<td>977</td>
<td>21,523</td>
<td>445</td>
<td>0.9684</td>
</tr>
</tbody>
</table>

#### Testing

<table>
<thead>
<tr>
<th>Hidden Layers</th>
<th>Neurons</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>ACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[50 50]</td>
<td>2,368</td>
<td>161</td>
<td>2,339</td>
<td>132</td>
<td>0.9414</td>
</tr>
<tr>
<td>2</td>
<td>[100 100]</td>
<td>2,391</td>
<td>185</td>
<td>2,315</td>
<td>109</td>
<td>0.9412</td>
</tr>
<tr>
<td>3</td>
<td>[50 50 50]</td>
<td>2,396</td>
<td>170</td>
<td>2,330</td>
<td>104</td>
<td>0.9452</td>
</tr>
<tr>
<td>3</td>
<td>[100 100 100]</td>
<td>2,396</td>
<td>114</td>
<td>2,386</td>
<td>104</td>
<td>0.9564</td>
</tr>
<tr>
<td>4</td>
<td>[100 100 100 100]</td>
<td>2,408</td>
<td>147</td>
<td>2,353</td>
<td>92</td>
<td>0.9550</td>
</tr>
<tr>
<td>5</td>
<td>[100 100 100 100 100]</td>
<td>2,384</td>
<td>128</td>
<td>2,372</td>
<td>116</td>
<td>0.9512</td>
</tr>
</tbody>
</table>

### TABLE IV
**Comparisons Between Deep Learning and Other Shallow Learning Based Classification Methods**

#### Training

<table>
<thead>
<tr>
<th>Method</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>ACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>21,338</td>
<td>1,781</td>
<td>20,719</td>
<td>1,162</td>
<td>0.9346</td>
</tr>
<tr>
<td>SVM</td>
<td>21,610</td>
<td>1,576</td>
<td>20,924</td>
<td>890</td>
<td>0.9452</td>
</tr>
<tr>
<td>NB</td>
<td>21,532</td>
<td>9,940</td>
<td>12,560</td>
<td>968</td>
<td>0.7576</td>
</tr>
<tr>
<td>DT</td>
<td>21,630</td>
<td>1,560</td>
<td>20,940</td>
<td>870</td>
<td>0.9460</td>
</tr>
<tr>
<td>DL4MD</td>
<td>22,035</td>
<td>953</td>
<td>21,547</td>
<td>465</td>
<td>0.9685</td>
</tr>
</tbody>
</table>

#### Testing

<table>
<thead>
<tr>
<th>Method</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>ACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>2,264</td>
<td>163</td>
<td>2,337</td>
<td>236</td>
<td>0.9202</td>
</tr>
<tr>
<td>SVM</td>
<td>2,305</td>
<td>152</td>
<td>2,348</td>
<td>195</td>
<td>0.9306</td>
</tr>
<tr>
<td>NB</td>
<td>2,332</td>
<td>1,541</td>
<td>959</td>
<td>168</td>
<td>0.6582</td>
</tr>
<tr>
<td>DT</td>
<td>2,357</td>
<td>292</td>
<td>2,208</td>
<td>143</td>
<td>0.9130</td>
</tr>
<tr>
<td>DL4MD</td>
<td>2,396</td>
<td>114</td>
<td>2,386</td>
<td>104</td>
<td>0.9564</td>
</tr>
</tbody>
</table>
6 Conclusion and future work

In this paper, based on Windows API calls extracted from a real and large file sample collection, we design a deep learning framework using the SAEs model for malware detection, which consists of two phases: unsupervised pre-training and supervised backpropagation. The SAEs model performs as unsupervised pre-training in a greedy layer-wise fashion putting the parameters of all layers in a region of parameter space in a bottom-up way, and then supervised BP is adopted to tune the multilayer model’s parameters in a top-down direction. To the best of our knowledge, this is the first work investigating deep learning using the SAEs model based on Windows API calls in malware detection for real industrial application. A comprehensive experimental study on a real and large file collection from Comodo Cloud Security Center is performed to compare various malware detection approaches. The experimental results obtained demonstrate that our proposed method can further improve the overall performance in malware detection compared with traditional shallow learning methods and that deep learning is a viable architecture for malware detection.

In our future work, we will further explore how sparsity constraints are imposed on AutoEncoder and how sparse SAEs can be designed to further improve malware detection effectiveness. Meanwhile, it would be interesting to investigate other deep learning models for malware detection.

Acknowledgment

The authors would also like to thank the anti-malware experts of Comodo Security Lab for the data collection as well as helpful discussions and supports. This work is partially supported by the U.S. National Science Foundation under grant CNS-1618629.

References


