Unsupervised intrusion detection through skip-gram models of network behavior

Rafael San Miguel Carrasco, Miguel-Angel Sicilia
University of Alcalá, Madrid, Spain

Keywords

Skip-gram modeling, **Neural networks, Anomaly detection,** Intrusion detection, Unsupervised learning, Word2vec, Unknown attacks

Abstract

Detecting intrusions is one of the main objectives of computer security. Attacks have become overly sophisticated over the years in order to remain effective and stealthy. Major breaches are typically perpetrated using techniques that are polymorphic, multi-vector, multi-stage and targeted, that is, adopting forms that were never seen before. Anomaly detection, which doesn't make any assumption about the shape of a potential attack but instead on legitimate behavior, seems to be a suitable approach in order to defeat sophisticated intrusions. Skip-gram modeling, a word2vec algorithm variant, was leveraged to model systems' legitimate network behavior. The resulting model was then used to spot intrusions in a test dataset. The optimal configuration led to 99,20% precision, 82,07% recall, and 91,02% accuracy, with a false positive rate of 0,61%, which is significantly lower than most state-of-the-art methods. These metrics were achieved under a fully unsupervised setting, that is, without any prior knowledge of what constitutes an attack. Furthermore, the approach provides benefits in terms of interpretability and log storage requirements, as it requires a small amount of input features. It also produces information about systems behavior and their relationships, that can be reused by other analysis techniques to obtain further insights.

1. Introduction

Computer and network intrusion detection is a popular research field. One of their objectives is to design and implement intrusion detection techniques with high detection rates and low false positive rates [1]. As cyber attacks become more sophisticated, the main challenge becomes deploying techniques that can recognize unknown attacks [2], that is, those for which an statistical or signature-based pattern doesn't exist. Another related issue is the need to handle huge amounts of data that are collected [3] from multiple data sources, and the fact that not all data might be relevant and relevant data might not be present. Multiple general-purpose algorithms have been re-engineered for intrusion detection. They fall into one of these categories: misuse detection and anomaly detection.

Misuse detection detects attacks by comparing current activity with the expected actions of an attacker [4]. In contrast, anomaly detection builds a normal activity profile for a system [5]. Misuse detection requires training observations to be labeled as normal or malicious activity. In anomaly detection, only legitimate behavior is considered during training. Behavior that significantly deviates from modeled legitimate behavior in the test data is then flagged as an anomaly. Labeled data for misuse detection might not be easy to obtain for the following reasons:

- A record might or might not constitute an attack depending on the context. Likewise, an attack might relate to multiple records. These relationships cannot be captured by record-level labels.
- Related records required to recognize an attack might have been triggered by different data sources.

- No datasets containing real-life traffic are generally available. Moreover, cyberattacks rates and impact metrics remain unavailable to researchers [6].

In contrast, anomaly detection focuses on finding patterns in data that don't conform to expected behavior [7]. It is suitable for the intrusion detection field for the following reasons:

- Evading anomaly detection mechanisms might be much harder than traditional signature-based systems for an attacker.
- Technology diversity doesn't interfere with general principles used in anomaly detection.
- Anomaly detection is generally resilient to new threat vectors, because the availability of a new vector doesn't change the expected legitimate behavior of a system or network vulnerable to that vector.

Previous work [8] identified the following pros of anomaly detection: ability to detect unknown attacks, lower dependency on OS, and accurate detections of privilege abuse. However, building profiles from data is not trivial [9], given huge network traffic volumes, highly imbalanced attack class distribution, complex decision boundaries to distinguish normal and abnormal behavior, and the need to adapt to a constantly changing environment that is therefore subject to concept drift [10].

The cons of anomaly detection [8] can be summarized as follows: alerts not being triggered on time, unavailability when updating normal profiles, and profiles becoming obsolete due to ever-changing network traffic. It might also be vulnerable to attacks against profile learning features [11]. However, these techniques rely on prior knowledge about the algorithm and the possibility to inject data during the training phase. Lastly, training data might contain malicious actions, and legitimate activity might not be frequent [12].

Our work has focused on developing a technique that leverages the pros of anomaly detection. Particularly, we have designed, implemented and tested an approach based on skip-gram modeling that learns the relationships among the entities of a network and their respective behavior.

The information learnt in the training process allows to measure the degree of departure from expected behavior and spot malicious activity in the detection phase. This approach is has low log storage requirements, as it requires just four network-related features to be modeled. Compared to other intrusion detection techniques [43] that require the 49 features of the UNSW-NB15 dataset, our approach requires 82,7% less storage to achieve comparable results. It's also interpretable, and it provides insights that can be exploited by other techniques, such as Peer Group Analysis.

Tested against the UNSW-NB15 dataset [13], it achived 99,20% precision and 82,07% recall.

The rest of this paper is structured as follows. In Section 2, previous related research is summarized, and the skip-gram modeling is presented, along with a number of successful Use Cases in other fields. In Section 3, the methodology followed to re-engineer the skip-gram modeling algorithm for intrusion detection is explained. Section 4 covers our experimental setup to test the effectiveness of the modified algorithm. Finally, Section 5 includes a discussion of the results and relevant conclusions.

2. Background

Here we present previous research directly related to the approach reported later, and then cover the fundamentals of the techniques applied.

2.1. Previous research

Multiple algorithm types and techniques have already been used to implement anomaly detection for intrusion detection, with different degrees of success. Recent surveys [1] [8] [53] have developed taxonomies of these techniques and also describe them in detail, referencing the related papers and comparing their performance. Other surveys [7] on anomaly detection have put focus on successful applications to the intrusion detection domain.

From the techniques described in the aforementioned surveys, those implemented through neural networks are the ones receiving further attention in the research community [14].

Self-Organizing Maps (SOM), that provide an neural-network based clustering algorithm, have proven to effectively detect intrusions by modeling legitimate behavior, under multiple scenarios [15] [16] [17] [18]. Tested against the 41 features of the KDD dataset, it achieved a false positive rate of 1.38% and detection rate of 90.4%

[19]. Adaptive Resonance Theory (ART) is also a neural network-based clustering technique whose underlying logic is very similar to SOM. From 27 features extracted from IP, TCP, UDP and ICMP headers of a network traffic dataset, with 27 different attack types, an ART-based IDS obtained 97% accuracy, while the SOM-based IDS achieved 95% [20].

Recurrent Neural Networks (RNN) can learn sequences of normal user behavior during a session, and predict the most probable next action afterwards [21]. Legitimate behavior leads to accurate predictions, while high prediction errors are a signal of anomalous behavior.

Recent research [22] has leveraged a neural network-based NLP (Natural Language Processing) algorithm for intrusion detection. This research was based on Paragraph Vector algorithm, that produces document-level embeddings, as opposed to the word embeddings used in our work, that are easier to compute. The proposed method targets C&C (command-and-control) detection, that is, malware communications, and it requires sample malicious traffic to train a SVM (Support Vector Machine) classifier. The method achieved varying values of precision (0.77-0.99) and recall (0.69-1.00), depending on selected training method and test data. Authors state effectiveness relies on having good training data, which is a strong requirement in any supervised setting. Our proposed approach is based on an unsupervised setting instead. Therefore, prior knowledge about intrusion tactics is not required to make it effective.

Finally, there have been attempts to perform intrusion detection with neural networks under supervised settings, that is, by classifying instances based on their labels or otherwise including attack traffic in the training phase. Convolutional Neural Networks (CNNs) [23] have been used to detect malicious URLs, file paths and registry keys, with 97,8%-99,3% detection rates. CMAC (Cerebellar Model Articulation Controller) neural network [24] has been combined with Reinforcement Learning feedback to detect variants of DoS attacks after training the system with a given DoS attack type and benign traffic. The system achieved an error of 3,28E-05% when identifying *a priori* attack patterns.

2.2. Skip-gram modeling

Word2vec is a technique used for learning vector representations of words, or embeddings. There are two variants of word2vec: Continuous Bag-of-Words (CBOW), that predicts a target word from its context words, and skip-gram, that predicts context words from a given target word. The loss function is minimized when the model produces high probabilities for words that are part of the target word's context, and low probabilities to other words (noise).

The differences between CBOW and skip-grams is that the latter works better for large datasets because it treats each context-target pair as a single data point. Therefore, skip-gram is more suitable in the context of intrusion detection, where large security log files are typically required to be processed in order to spot attacks. Extensions to the original algorithm were added [26] after the original paper was published to improve the quality of the vectors and the training speed. Related words have vectors with a high cosine similarity, and a low similarity otherwise. It's therefore able to accurately predict the context of a word after the training process has been completed.

In our research, skip-gram algorithm has been re-engineered to make it suitable for network intrusion detection, under the initial hypothesis that systems and their network connections can be effectively represented as words from a text for behavior modeling purposes.

2.3. Existing Use Cases

Both word2vec and skip-gram modeling have been very recently applied to several use cases outside the computer security field. For instance, in the medical field, skip-gram has proven to help in finding relationships between cancer patents [27], as well as modeling medical words to encode its semantic information [28]. In biology, it has been used for protein classification [29]. In the marketing field, it has been used in food sensory evaluation analysis for predicting consumer acceptance [30], and for sentiment analysis from movie reviews [31]. Other use cases include modeling harmony [32], human judgment evaluation [33], and psycholinguistic properties inference [34].

Skip-gram has also been leveraged in the anomaly detection field, particularly to spot deviations from normal state in log files [47]. In the computer security field, word2vec was used as a feature engineering tool, training classifiers with the word embeddings generated for both normal traffic and intrusion instances [48]. A similar

approach has been proposed to find the semantics of particular attack types, which could then be fed as input to classifiers as Convolutional Neural Networks [49].

3. Methodology

3.1. Dataset description

The dataset used for our research is the UNSW-NB 15 dataset [13]. Data was created with the security product IXIA PerfectStorm [35] in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS). The objective was to generate a hybrid of real modern normal activities and synthetic contemporary attack behaviours.

This dataset has been reported to address several issues found in the KDD99 and NSLKDD datasets. Particularly those regarding obsolete attack types, obsolete legitimate traffic scenarios, and the unbalanced of training and testing instances [36]. Other datasets like IDEVAL have also been heavily criticized [37], despite still being very commonly used to measure anomaly detectors' performance.

In UNSW-NB15, the tcpdump command was used to capture 100 GB of raw traffic in PCAP format. The dataset contains nine attack categories: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. Argus, Bro-IDS and a set of twelve custom algorithms were used to generate 49 features that are enumerated in Annex A. The class label indicates whether the record is normal traffic or an attack. The dataset contains 2,540,044 records stored in 4 CSV files: UNSW-NB15_1.csv, UNSW-NB15_2.csv, UNSW-NB15_3.csv and UNSW-NB15_4.csv. These were first merged into a single file to generate the required training and test dataset.

3.2. Feature engineering

Our features selection criteria relied on the general assumption that in order to perpetrate an attack, an intruder that has compromised a system must access systems and services in a way that differs from normal behavior seen before.

From the 49 features available in the dataset, only four were required for our skip-gram modeling purposes. These are the four main variables that characterize network accesses: source IP address (srcip), target IP address (dstip), target port (dport) and network protocol (proto). The remaining features have been discarded for the reasons stated below:

- Features derived from aggregations of raw features. These features are expensive to compute in a production environment, and including them would also prevent real-time detection.
- Protocol-based features. These features are not available from many network-related data sources, and so including them would make our approach less generally applicable.
- Application-based features. These features, if included in the analysis, would limit the scope of detection to the related applications, preventing the algorithm to generalize.
- Time-based features. An attacker could easily imitate time-related behavior of legitimate users, and so these values can't reliably characterize malicious activity.

UNSW-NB15 features are enumerated in Appendix A in order to let readers understand how the aforementioned reasons apply to the related features.

In general, our aim is to avoid developing a technique that relies on features not generally available or whose synthesis process is not either scalable or representative of real-world datasets. This leads to lower log storage requirements to deploy the algorithm, compared to other techniques that require all dataset features [43]. Particularly, the storage requirements drop from 586,4MB (49 features) to 101,2MB (4 features plus the label), that is, 82,7% less. This feature addresses one of the key issues of detecting intrusions [50]: the need to handle huge amounts of data, where not all data might be relevant.

Target port feature was simplified to only hold values of most popular ports, or a default value (arbitrarily set to 9999) for any other port. This approach serves the purpose of letting the algorithm focus on what is more relevant for network behavior modeling: whether a system is connecting to a well-known or uncommon service. This assumption was later confirmed by running the skip-gram modeling exercise against the original and the extracted features and comparing the results. Using the simplified *dport* feature led to higher accuracy.

3.3. Neural network design

TensorFlow [38] is an open-source framework developed by Google that allows to design and execute machine-learning algorithms. It has been used for research across multiple fields: speech recognition, robotics, computer vision, information retrieval, and NLP, among others. In our research, TensorFlow was used to implement skip-gram modeling through a neural network. The neural network architecture is composed of three layers: an input layer, an embeddings layer and a softmax classifier that produces the output.

This is visually depicted in the following figure:

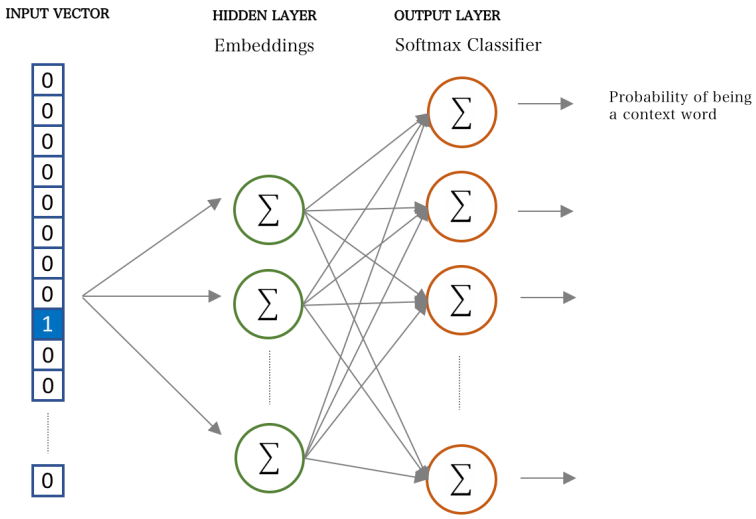

Figure 1. Skip-gram neural network architecture.

The matrix of embeddings is randomly initialized with values between -1 and 1 that are drawn from a uniform distribution. The loss is defined in terms of a logistic regression model leveraging negative sampling. Therefore, a matrix of weights and biases is also defined and initialized. The biases are initialized to zero, while the weights are initialized with values draw from a normal distribution. To our knowledge, no other network architecture has proven to perform better for skip-gram modeling. In any case, the design is still subject to parameter fine-tuning, which has been implemented in our setting through grid search, as explained in Section 4. We have also redesigned the microbatch function, as explained in Section 3.5, to ensure that inputs are provided in a format suitable for a NLP neural network like this.

Stochastic Gradient Descent (SGD) is used to update weights. Particularly, Adagrad optimizer was chosen, as it dynamically adjusts the learning rate according to how frequent a given parameter is. This is the best choice for sparse data like the one hot encoding vectors used in skip-gram. After all epochs have been run, the normalized embeddings, suitable for calculating cosine similarity between entities, are returned.

3.4. Re-engineering approach

word2vec implementations are designed to work with words in a text. In the neural network implementation of this algorithm, microbatches containing pairs of a target word and some of its context words are repeatedly generated in the training phase in order to learn progressively more accurate embeddings reflecting existing relationships. The end goal is being able to accurately predict what words belong to the context of a given word, and which don't, from what it's called a negative sample. This is shown in the following figure.

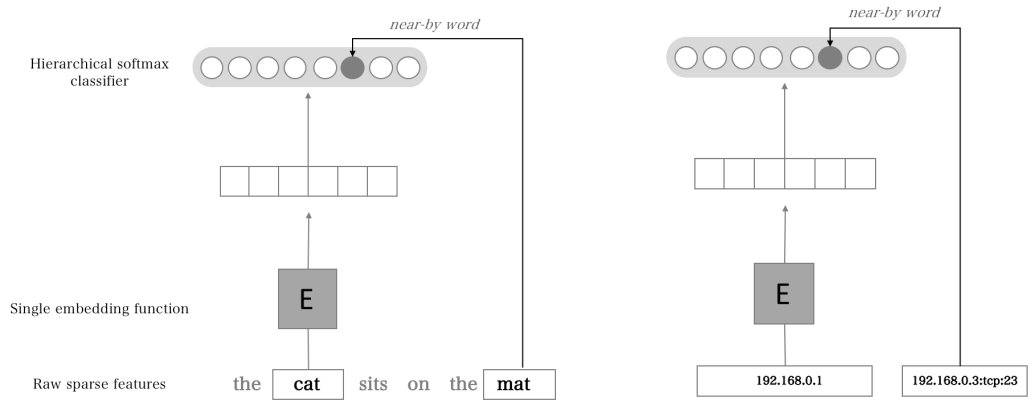

Figure 2. Skip-gram modeling goal in text classification and in intrusion detection.

In our setting, aimed at network behavior modeling, target words represent *systems* that initiate network connections, while words related to them (context) represent *connection types*, that are characterized by a destination IP address, a network protocol and a target port. A given connection belongs to a connection type if its target IP address, port and protocol matches the values for that connection type.

In this scenario, context (i.e. related words) in which a given target word (system) is found represents the most frequent connection types (target IP, port, protocol) initiated by that system. Therefore, the learning process aims at obtaining locations (numerical representations) in a hyperplane of both systems and connection types, that satisfy the following three conditions:

1) Systems that are close in the hyperplane connect to similar IP addresses and ports.
2) If a system is close to a connection, it means that that type of connection is frequent for that system.
3) If two or more connections are close, it means that they are initiated by the same set of systems.

In other words, short distances represent high similarity, that is, two systems with similar behavior or a connection type that is frequently initiated by a system. Likewise, long distances can either represent connection types that are unlikely to be initiated by a system, that is, connections that are no expected for that system, or systems whose behavior is very different.

In the training phase, embeddings are calculated and recorded. In the detection phase, each event is processed by obtaining the distance between the embedding of the system initiating the connection and the embedding of the connection profile used. This distance can be used to discover when a system is deviating from expected behavior. Multiple connections initiated by a system translating into long distances constitute a reliable indicator of an anomalous behavior.

This is represented in the following figure:

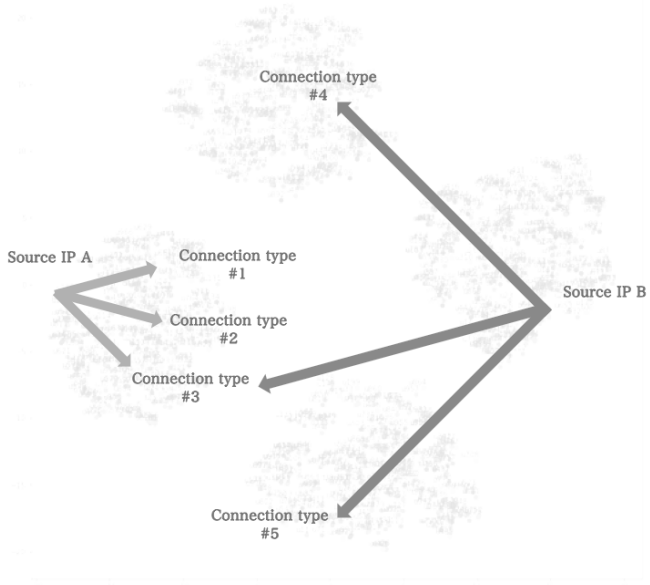

Figure 3. Examples of normal and anomalous interactions.

On the other hand, producing numerical representations of systems and connection types reduces false positive rate, because the likelihood of a connection type being initiated by a given system doesn't only depend on the behavior of that system in the training phase, but also on the behavior of other systems with which it exhibits a high similarity. In other words, the behavior profile considered legitimate for a system is *inferred* from observed behavior *and* the behavior of similar systems.

Therefore, if some legitimate behavior is not captured during the training phase for a system, but it is for related systems, that behavior won't be considered anomalous in the detection phase.

3.5. Microbatch building function

For the skip-gram modeling process to work for our network traffic dataset, the microbatch generation function originally designed for NLP tasks had to be reengineered. Each record in our dataset is a list of items, whose first item represents a system, and the rest are connection types observed for that system in the training data. Multiple instances of the same connection type can be found in each record if the system has initiated multiple connections of that type in the training period. In other words, there are as many occurrences of a connection type in a system's record as connections of that type were observed in the training data. Therefore, each record encapsulates all the connection types seen for a given system, and the number of times that they have been seen.

Our redesigned microbatch function runs through all the records by taking consecutive non-overlapping samples of the dataset, as determined by *batch_size* parameter. For each record, it produces a target word and a set of related words (context). The target word can be a system ($p = 0.8$) or a connection type ($p = 0.2$). In the first case, the objective is to learn the relationship between a system and its connection types. In the latter case, the objective is to learn the relationship between a connection type seen for a given system and other connection types also seen for that system. This stochastic feature was introduced because it led to more accurate embeddings.

In both cases, context is comprised of connection types found in that record, randomly selected. The number of items picked up to build the context is determined by the *num_skips* parameter, which represents the *bandwidth* or *depth* of the context.

3.6. Distance measurement

Skip-gram produces a vectorial representation of a word (system or connection type) that allows to find related words, which are those whose distance to the word in the hyperplane is short. The measure of distance produced by our skip-gram modeling implementation is cosine similarity.

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine similarity values fall into the interval [-1, 1], with 1 being the highest similarity (shortest distance) and -1 being the lowest (longest distance). Vectors representing related words will have an angle close to 0 degrees (i.e. cosine similarity close to 1), and those of unrelated words will have an angle between 90 and 180 degrees (i.e. cosine less than 0). Cosine similarity is calculated for all events processed in the detection phase.

In our implementation, the similarity formula is translated into a matrix multiplication operation. The normalized embeddings of the systems are multiplied by the trasposed embeddings of the connection profiles, resulting in scalar products that produce a similarity value for each event.

3.7. Visual inspection technique

Formal performance measurement criteria were defined to check the accuracy of our proposed technique. However, before getting to that final stage, it's useful to inspect the results of the skip-gram modeling exercise. This inspection also allows to gain a better understading of data structure. Particularly, visual inspection is used to confirm whether clusters of systems and connection profiles are formed. That would be the expected outcome, because systems in a network tend to behave according to the role of the user that uses them (workstations), or according to the businnes function assigned to them (servers).

For this purpose, the obtained embeddings, whose number of dimensions ranges from 4 to 16, are transformed into vectors of 2 dimensions using tSNE, so that they can be plotted in the screen. t-Distributed Stochastic Neighbor Embedding (tSNE) is a dimensionality reduction technique used for visualization of high-dimensional datasets [39] [40] [41]. It's made available to Python environments through the *scikit-learn* package. The parameters used for plotting the embeddings are shown in Table 1. The resulting scatter plot obtained for the optimal configuration (C1) of the neural network is depicted in Figure 4. The plot provides support to the hypothesis that there are systems in the network performing similar activities.

Table 1. tSNE parameters.

Parameter	Value
perplexity	50
n_components	2
init	pca
n_iter	5000

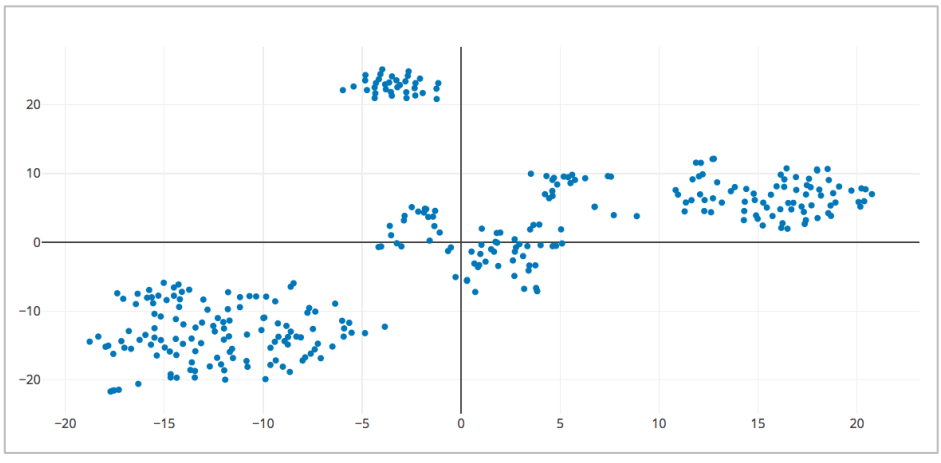

Figure 4. Scatter plot of obtained embeddings.

3.8. Performance evaluation criteria

Our performance evaluation criteria is based on the most common performance metrics in the intrusion detection field: precision and recall [42]. These are the related formulas:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

The F-score, also popular in this domain, can be calculated from precision and recall values:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

True positives (*tp*) are events classified as an intrusion that are an indeed an intrusion. Likewise, false positives (*fp*) are events classified as intrusions that are legitimate traffic, and false negatives (*fn*) are intrusions classified as legitimate traffic. Precision measures the proportion of true positives detected among the events classified as intrusions, while recall measures the proportion of intrusions detected among the total events.

4. Experimental setup

4.1. Training and test datasets

The full UNSW-NB15 dataset contained 2,540,044 events. It was produced by merging the content of these four files: UNSW-NB15_1.csv, UNSW-NB15_2.csv, UNSW-NB15_3.csv, and UNSW-NB15_4.csv, that can be downloaded from the following URL:

https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/

1,938,119 events (76,3%) were sampled from the full dataset to train the neural network. The training dataset contained no attacks, as expected under an unsupervised paradigm setting. 601,928 events were sampled from the full dataset to test the accuracy of the model, which represents approximately 23,6% of the full dataset.

280,645 events in the test dataset were attack-free, and 321,283 events represented actual attacks, belonging to the aforementioned attack categories.

4.2. Parametrization

Grid search was used to find the optimal parameters for the neural network. In our context, optimal configurations is defined as the one having higher precision and higher recall, with recall having higher priority. The following table describes the relevant parameters of the neural network.

Table 2. Neural network parameters.

Parameter	Value
batch_size	Number of records supplied by the batch generation function to the neural network in each epoch.
num_skips	Number of words included in the context of a given target word when generating a micro batch.
valid_size	Number of records used for validation.
num_steps	Number of epochs.
embedding_size	Number of dimensions of the vector representing each word.

4.3. Results and discussion

In our setting, an event was considered to represent an attack if the distance (cosine similarity) from the system to the connection profile shown in the event was lower than zero. Likewise, an event was considered to represent legitimate traffic if the cosine similarity value was zero or positive. 2,830 unique combinations of system

and connection profile were extracted from the test dataset and classified as either normal or attack. The following table shows the results obtained with each configuration. The following information is included:

- Configuration parameters
  - batch_size
  - num_skips
  - num_steps
  - embedding_size
- Number of true positives, negatives and also false positives and negatives.
- Average loss reported by TensorFlow after running the last epoch
- Lastly, precision and recall metrics, used for performance evaluation

	batch_size	num_skips	num_steps	embedding_size	True positives	False positives	True negatives	False negatives	% of legitimate traffic (training)	Average loss	Precision	Recall
C1	**8**	**2**	**90,000**	**4**	**1,122**	**9**	**1,454**	**245**	**98.14**	**0.197208**	**0,9920**	**0,8207**
C2	16	2	90,000	4	1,103	7	1,456	264	99.01	0.360984	0,9936	0,8068
C3	32	2	90,000	4	1,109	8	1,455	258	99.18	0.626274	0,9928	0,8112
C4	16	4	90,000	4	1,114	7	1,456	253	98.48	0.364535	0,9937	0,8149
C5	16	4	90,000	8	1,119	9	1,454	248	97.55	0.367023	0,9920	0,8185
C6	16	4	90,000	16	1,118	14	1,449	249	97.79	0.366863	0,9876	0,8178
C7	16	4	180,000	4	1,103	6	1,457	264	98.89	0.355693	0,9945	0,8068
C8	16	4	360,000	4	1,096	7	1,456	271	99.30	0.359761	0,9936	0,8017
C9	32	4	360,000	8	1,088	5	1,458	279	99.30	0.624516	0,9954	0,7959

Table 3. Configuration parameters and results.

Table 3 shows that bigger batch sizes, skips lengths or embedding sizes didn't significantly influence the algorithm's performance. Also, the algorithm quickly achieved convergence: increasing the number of epochs didn't lead to better performance. This circumstance might not hold for bigger datasets though.

According to our performance evaluation criteria, C1 was the optimal configuration, with 99,20% precision and 82,07% recall (detection rate), leading to an F-score of 89,82%. The accuracy was 91,02%, and the false positive rate 0,61%. Higher precisions were obtained by other configurations, but at the expense of a higher recall. These metrics were achieved without any prior knowledge of what constitutes an attack.

Other researchers have also tested their intrusion detection technique against UNSW-NB15 dataset [43]. The following table summarizes how our proposal compares to other supervised and unsupervised techniques based on neural network implementations [1]:

Technique	Dataset	Requires prior knowledge	False positive rate	Accuracy	Detection rate	
Skip-gram	**UNSW-NB15**	**No**	**0,61%**	**91,02%**	**82,07%**	
MLP-BP	KDD Cup'99	Yes	**8,51%**	-	81,96%	
RBF	KDD Cup'99	Yes	**1,2%**	-	99,2%	
SOM	KDD Cup'99	No	**1,38%**	-	90,4%	

ART	KDD Cup'99	No	**3,86%**	-	96,13%	
LSTM-RNN	KDD Cup'99	No	-	96,93%	98,88%	
DBN	KDD Cup'99	Yes	**0,76%**	93,49%	-	
Ramp-KSVCR	UNSW-NB15	Yes	**2,46%**	-	98,68%	

Table 4. Performance comparison with other neural network-based techniques.

Our approach outperforms all these techniques in terms of false positive rate, which remains to be one of the main challenges in intrusion detection systems [44], while still providing a comparable accuracy and an acceptable detection rate. It does so without any prior knowledge about the attacks and using a small amount of network-related inputs that are available from any network device.

Also, our approach has been tested against a recent dataset (2015) that doesn't suffer from the issues found [51][52] in datasets from 1999 used by other techniques in Table 4. These issues include: redundant records in the test dataset leading to biased evaluation results [51], and the fact that randomly selecting subsets of the training and testing data allows to achieve unrealistically high accuracies [52].

5. Conclusions and outlook

The optimal configuration of our proposed intrusion detection algorithm led to 99,20% precision, 82,07% recall, and 91,02% accuracy, with a false positive rate of 0,61%, which is significantly lower than most state-of-the-art methods. These metrics were achieved under a fully unsupervised setting, that is, without any prior knowledge of what constitutes an attack.

Furthermore, the proposed algorithm makes several contributions. First, it requires a small number of features that can be easily obtained from a network or security sensor in order to accurately model systems behavior.

This in turns results in less storage required to store the data and run the algorithm. Also, results can easily be interpreted. The reason why an event was classified as an attack can be explained in terms of how infrequent that behavior is, not only for that system, but also for systems exhibiting a similar behavior. Lastly, the model shows what clusters of related systems behavior exist, which can be used as input to further techniques and analyses, particularly those that leverage Peer Group Analysis [45] [46].

Appendix A. UNSW-NB 15 dataset feature set

Name	Description
srcip	Source IP address
sport	Source port number
dstip	Destination IP address
dport	Destination port number
proto	Transaction protocol
state	State. Protocol-dependant. Examples: ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used)
dur	Total duration
sbytes	Source to destination transaction bytes
dbytes	Destination to source transaction bytes
sttl	Source to destination Time To Live (TTL)
dttl	Destination to source Time To Live (TTL)
sloss	Source packets retransmitted or dropped
dloss	Target packets retransmitted or dropped
service	http, ftp, smtp, ssh, dns, ftp-data ,irc, and (-) (unusual service)
sload	Source bits per second
dload	Destination bits per second
spkts	Source to destination packet count
dpkts	Destination to source packet count
swin	Source TCP window advertisement value
dwin	Target TCP window advertisement value
stcpb	Source TCP base sequence number
dtcpb	Destination TCP base sequence number
smeansz	Mean packet size transmitted by the src
dmeansz	Mean packet size transmitted by the dst
trans_depth	Pipelined depth into the connection of HTTP request/response transaction
res_bdy_len	Size of uncompressed data transferred from the server’s HTTP service
sjit	Source jitter (ms)
djit	Destination jitter (ms)
stime	Start time
ltime	Last time
sintpkt	Source interpacket arrival time (ms)
dintpkt	Destination interpacket arrival time (ms)
tcprtt	TCP connection setup round-trip time: sum of ’synack’ and ’ackdat’
synack	TCP connection setup time: time between SYN and SYN_ACK packets
ackdat	TCP connection setup time: time between SYN_ACK and ACK packets
is_sm_ips_ports	1 if source and destination IP addresses and ports equal, 0 otherwise
ct_state_ttl	Number for each state according to specific range of values for source/target TTL
ct_flw_http_mthd	Number of flows with methods such as GET and PORT in HTTP service

is_ftp_login	1 if FTP session is authenticated, 0 otherwise
ct_ftp_cmd	Number of flows with a command in the FTP session
ct_srv_src	Connections that contain the same service and source IP address in 100 connections, according to last time
ct_srv_dst	Connections that contain the same service and destination IP address in 100 connections, according to last time
ct_dst_ltm	Connections with the same destination IP address in 100 connections, according to last time
ct_src_ ltm	Connections with the same source IP address in 100 connections, according to last time
ct_src_dport_ltm	Connections with the same source IP address and destination port in 100 connections, according to last time
ct_dst_sport_ltm	Connections with the same destination IP address and source port in 100 connections, according to last time
ct_dst_src_ltm	Connections with the same source and destination and IP address in 100 connnections, according to last time
attack_cat	Attack category: Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
label	0 for normal traffic and 1 for attacks

References

[1] Hodo, E., Bellekens, X., Hamilton, A., Tachtatzis, C., & Atkinson, R. (2017). Shallow and deep networks intrusion detection system: A taxonomy and survey. *arXiv preprint arXiv:1701.02145*.

[2] Casas, P., Mazel, J., & Owezarski, P. (2012). Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, *35*(7), 772-783.

[3] Garvey, T. D., & Lunt, T. F. (1991, October). Model-based intrusion detection. In *Proceedings of the 14th national computer security conference* (Vol. 17).

[4] Cannady, J. (1998, October). Artificial neural networks for misuse detection. In *National information systems security conference* (Vol. 26).

[5] Chimphlee, W., Abdullah, A. H., Sap, M. N. M., Srinoy, S., & Chimphlee, S. (2006, November). Anomaly-based intrusion detection using fuzzy rough clustering. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on* (Vol. 1, pp. 329-334). IEEE.

[6] Kuypers, M. A., Maillart, T., & Pate-Cornell, E. (2016). An empirical analysis of cyber security incidents at a large organization. *Department of Management Science and Engineering, Stanford University, School of Information, UC Berkeley, http://fsi. stanford. edu/sites/default/files/kuypersweis_v7. pdf, accessed July*, *30*.

[7] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 15.

[8] Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16-24.

[9] Ye, N. (2000, June). A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop* (Vol. 166, p. 169). West Point, NY.

[10] Lane, T., & Brodley, C. E. (1998, August). Approaches to Online Learning and Concept Drift for User Identification in Computer Security. In *KDD* (pp. 259-263).

[11] Chebrolu, S., Abraham, A., & Thomas, J. P. (2005). Feature deduction and ensemble design of intrusion detection systems. *Computers & security*, *24*(4), 295-307.

[12] Estevez-Tapiador, J. M., Garcia-Teodoro, P., & Diaz-Verdejo, J. E. (2004). Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications*, *27*(16), 1569-1584.

[13] Moustafa, N., & Slay, J. (2015, November). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015* (pp. 1-6). IEEE.

[14] Choksi, K., Shah, B., & Kale, O. (2014). Intrusion detection system using self organizing map: a surevey. *Int. J. Engin. Res. Appl*, *4*(12), 11-16.

[15] Kumar, V. D., & Radhakrishnan, S. (2014, April). Intrusion detection in MANET using self organizing map (SOM). In *Recent Trends in Information Technology (ICRTIT), 2014 International Conference on* (pp. 1-8). IEEE.

[16] Sarasamma, S. T., Zhu, Q. A., & Huff, J. (2005). Hierarchical Kohonenen net for anomaly detection in network security. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *35*(2), 302-312.

[17] Rhodes, B. C., Mahaffey, J. A., & Cannady, J. D. (2000, October). Multiple self-organizing maps for intrusion detection. In *Proceedings of the 23rd national information systems security conference* (pp. 16-19).

[18] Ramadas, M., Ostermann, S., & Tjaden, B. (2003, September). Detecting anomalous network traffic with self-organizing maps. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 36-54). Springer, Berlin, Heidelberg.

[19] Kayacik, H. G., Zincir-Heywood, A. N., & Heywood, M. I. (2007). A hierarchical SOM-based intrusion detection system. *Engineering applications of artificial intelligence*, *20*(4), 439-451.

[20] Amini, M., Jalili, R., & Shahriari, H. R. (2006). RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks. *Computers & Security*, *25*(6), 459-468.

[21] Debar, H., Becker, M., & Siboni, D. (1992, May). A neural network component for an intrusion detection system. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on* (pp. 240-250). IEEE.

[22] Mimura, M., & Tanaka, H. (2017, November). Reading Network Packets as a Natural Language for Intrusion Detection. In *International Conference on Information Security and Cryptology* (pp. 339-350). Springer, Cham.

[23] Saxe, J., & Berlin, K. (2017). eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys. *arXiv preprint arXiv:1702.08568*.

[24] Cannady, J. (2000, October). Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In *Proceedings of the 23rd national information systems security conference* (pp. 1-12).

[25] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[26] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

[27] Whitehead, M., & Johnson, D. K. (2017). A Tool for Visualizing and Exploring Relationships among Cancer-Related Patents.

[28] Zhou, Z., Fu, B., Qiu, H., Zhang, Y., & Liu, X. (2017, April). Modeling medical texts for distributed representations based on Skip-Gram model. In *Information Management (ICIM), 2017 3rd International Conference on* (pp. 279-283). IEEE.

[29] Islam, S. A., Heil, B. J., Kearney, C. M., & Baker, E. J. (2017). Protein classification using modified n-gram and skip-gram models. *bioRxiv*, 170407.

[30] Kim, A. Y., Ha, J. G., Choi, H., & Moon, H. (2018). Automated Text Analysis Based on Skip-Gram Model for Food Evaluation in Predicting Consumer Acceptance. *Computational Intelligence and Neuroscience*, *2018*.

[31] Chakraborty, K., Bhattacharyya, S., Bag, R., & Hassanien, A. E. (2018, February). Comparative Sentiment Analysis on a Set of Movie Reviews Using Deep Learning Approach. In *International Conference on Advanced Machine Learning Technologies and Applications* (pp. 311-318). Springer, Cham.

[32] Sears, D. R., Arzt, A., Frostel, H., Sonnleitner, R., & Widmer, G. (2017). Modeling Harmony with Skip-Grams. *arXiv preprint arXiv:1707.04457*.

[33] Hollis, G., Westbury, C., & Lefsrud, L. (2017). Extrapolating human judgments from skip-gram vector representations of word meaning. *Quarterly Journal of Experimental Psychology*, *70*(8), 1603-1619.

[34] Hartmann, N. S., Candido Jr, A., Paetzold, G. H., & Aluisio, S. M. (2017, September). A Lightweight Regression Method to Infer Psycholinguistic Properties for Brazilian Portuguese. In *Text, Speech, and Dialogue: 20th International Conference, TSD 2017, Prague, Czech Republic, August 27-31, 2017, Proceedings* (Vol. 10415, p. 281). Springer.

[35] "Ixia PerfectStorm," . Available: https://www.ixiacom.com/es/products/perfectstorm.

[36] Moustafa, N., & Slay, J. (2016). The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective*, *25*(1-3), 18-31.

[37] Mahoney, M. V., & Chan, P. K. (2003, September). An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 220-237). Springer, Berlin, Heidelberg.

[38] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[39] Van Der Maaten, L. (2014). Accelerating t-SNE using tree-based algorithms. *Journal of machine learning research*, *15*(1), 3221-3245.

[40] Van der Maaten, L., & Hinton, G. (2012). Visualizing non-metric similarities in multiple maps. *Machine learning*, *87*(1), 33-55.

[41] Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. *RBM*, *500*(500), 26.

[42] Wu, S. X., & Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied soft computing*, *10*(1), 1-35.

[43] Bamakan, S. M. H., Wang, H., & Shi, Y. (2017). Ramp loss K-Support Vector Classification-Regression; a robust and sparse multi-class approach to the intrusion detection problem. *Knowledge-Based Systems*, *126*, 113-126.

[44] Pietraszek, T. (2004, September). Using adaptive alert classification to reduce false positives in intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 102-124). Springer, Berlin, Heidelberg.

[45] Botros, S. M., Diep, T. A., & Izenson, M. D. (2004). "Method and apparatus for training a neural network model for use in computer network intrusion detection." *U.S. Patent No. 6,769,066*. Washington, DC: U.S. Patent and Trademark Office.

[46] Diep, T. A., Botros, S. M., & Izenson, M. D. (2003). "Features generation for use in computer network intrusion detection." *U.S. Patent No. 6,671,811*. Washington, DC: U.S. Patent and Trademark Office.

[47] Bertero, C., Roy, M., Sauvanaud, C., & Trédan, G. (2017, October). Experience Report: Log Mining using Natural Language Processing and Application to Anomaly Detection. In *28th International Symposium on Software Reliability Engineering (ISSRE 2017)* (p. 10p).

[48] Zhuo, X., Zhang, J., & Son, S. W. (2017, December). Network intrusion detection using word embeddings. In *Big Data (Big Data), 2017 IEEE International Conference on* (pp. 4686-4695). IEEE.

[49] Barot K., Zhang J., Woo Son S. (2016). Using Natural Language Processing Models for Understanding Network Anomalies. IEEE High Performance Extreme Computing Conference, Waltham, MA USA.

[50] Garvey, T. D., & Lunt, T. F. (1991, October). Model-based intrusion detection. In *Proceedings of the 14th national computer security conference* (Vol. 17).

[51] Gogoi, P., Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2012, August). Packet and flow based network intrusion dataset. In International Conference on Contemporary Computing (pp. 322-334). Springer, Berlin, Heidelberg.

[52] Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153-1176.

[53] Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., & Kim, K. J. (2017). A survey of deep learning-based network anomaly detection. *Cluster Computing*, 1-13.