The Workshops of the Thirty-Second
AAAI Conference on Artificial Intelligence

Recurrent Neural Network Language Models for Open Vocabulary Event-Level Cyber Anomaly Detection

Aaron Tuor,[1] **Ryan Baerwolf,**[2] **Nicolas Knowles,**[2]
Brian Hutchinson,[1,2] **Nicole Nichols**[1] **Robert Jasper**[1]
[1]Pacific Northwest National Laboratory
Richland, Washington
[2]Western Washington University
Bellingham, Washington

Abstract

Automated analysis methods are crucial aids for monitoring and defending a network to protect the sensitive or confidential data it hosts. This work introduces a flexible, powerful, and unsupervised approach to detecting anomalous behavior in computer and network logs; one that largely eliminates domain-dependent feature engineering employed by existing methods. By treating system logs as threads of interleaved "sentences" (event log lines) to train online unsupervised neural network language models, our approach provides an adaptive model of normal network behavior. We compare the effectiveness of both standard and bidirectional recurrent neural network language models at detecting malicious activity within network log data. Extending these models, we introduce a tiered recurrent architecture, which provides context by modeling sequences of users' actions over time. Compared to Isolation Forest and Principal Components Analysis, two popular anomaly detection algorithms, we observe superior performance on the Los Alamos National Laboratory Cyber Security dataset. For log-line-level red team detection, our best performing character-based model provides test set area under the receiver operator characteristic curve of 0.98, demonstrating the strong fine-grained anomaly detection performance of this approach on open vocabulary logging sources.

1 Introduction

To minimize cyber security risks, it is essential that organizations be able to rapidly detect and mitigate malicious activity on their computer networks. These threats can originate from a variety of sources including malware, phishing, port scanning, etc. Attacks can lead to unauthorized network access to perpetrate further damage such as theft of credentials, intellectual property, and other business sensitive information. In a typical scenario, cyber defenders and network administrators are tasked with sifting through vast amounts of data from various logging sources to assess potential security risks. Unfortunately, the amount of data for even a modestly-sized network can quickly grow beyond the ability of a single person or team to assess, leading to delayed response. The desire for automated assistance has and continues to encourage inter-domain research in cyber security and machine learning.

Signature-based approaches for automated detection can be highly effective for characterizing individual threats. Despite their high precision, they suffer from low recall and may fail to detect subtle mutations or novel attacks. Alternatively, given an unlabeled training set of typically benign activity logs, one can build a model of "normal behavior". During online joint training and evaluation of this model, patterns of normal usage will be reinforced and *atypical* malicious activity will stand out as anomalous. The features used to identify unusual behavior are typically statistical feature vectors associated with time slices, e.g., vectors of counts for types of activities taking place in a 24-hour window. Such systems developed in research have been criticized as brittle to differences in site-specific properties of real-world operational networks such as security constraints and variable usage patterns (Sommer and Paxson 2010).

The approach we introduce aims to minimize site-specific assumptions implicit in feature engineering, and effectively model variability in network usage by direct online learning of language models over log lines. Language models assign probabilities to sequences of tokens and are a core component of speech recognition, machine translation, and other language processing systems. Specifically, we explore the effectiveness of several recurrent neural network (RNN) language models for use in a network anomaly detection system. Our system dynamically updates the network language model each day based on the previous day's events. When the language model assigns a low probability to a log-line it is flagged as anomalous. There are several advantages to this approach:

1. **Reduced feature engineering:** Our model acts directly on raw string tokens, rather than hand-designed domain-specific statistics. This dramatically reduces the time to deployment, and makes it agnostic to the specific network or logging source configuration. It also removes the "blind spots" introduced when tens of thousands of log-lines are distilled down to a single aggregated feature vector, allowing our model to capture patterns that would have otherwise been lost.
2. **Fine grained assessment:** The response time for analysts can be improved by providing more specific and relevant events of interest. Baseline systems that alert to a user's day aggregate require sifting through tens of thousands of

actions. Our approach can provide log-line-level or even token-level scores to the analyst, helping them quickly locate the suspicious activity.

3. **Real time processing:** With the ability to process events in real time and fixed bounds on memory usage which do not grow over time, our approach is suitable for the common scenario in which log-line events are appearing in a high-volume, high-velocity log stream.

We assess our models using the publicly available Los Alamos National Laboratory (LANL) Cyber Security Dataset, which contains real (de-identified) data with ground truth red team attacks, and demonstrate language models definitively outperforming standard unsupervised anomaly detection approaches.

2 Prior work

Machine learning has been widely explored for network anomaly detection, with techniques such as isolation forest (Gavai et al. 2015; Liu, Ting, and Zhou 2008) and principal component analysis (Novakov et al. 2013; Ringberg et al. 2007) attracting significant interest. Machine learning classifiers ranging from decision trees to Naïve Bayes have been used for cyber security tasks such as malware detection, network intrusion, and insider threat detection. Extensive discussion of machine learning applications in cyber security is presented in (Bhattacharyya and Kalita 2013; Buczak and Guven 2016; Dua and Du 2016; Kumar, Kumar, and Sachdeva 2010; Zuech, Khoshgoftaar, and Wald 2015; Rubin-Delanchy, Lawson, and Heard 2016).

Deep learning approaches are also gaining adoption for specialized cyber defense tasks. In an early use of recurrent neural networks, Debar, Becker, and Siboni (1992) model sequences of Unix shell commands for network intrusion detection. Anomaly detection has been demonstrated using deep belief networks on the KDD Cup 1999 dataset (Alrawashdeh and Purdy 2016), and Bivens et al. (2002) use multi-layer perceptrons for the DARPA 1999 dataset. Both approaches use aggregated features and synthetic network data. Tuor et al. (2017) and Veeramachaneni et al. (2016) both employ deep neural network autoencoders for unsupervised network anomaly detection using time aggregated statistics as features.

Some works of note have been previously published on the LANL data. Turcotte, Heard and Kent (2016) develop an online statistical model for anomaly detection in network activity using Multinomial-Dirichlet models. Similarly, Turcotte et al. (2016) use Poission Factorization (Gopalan, Hofman, and Blei 2013) on the LANL authentication logs. A user/computer authentication count matrix is constructed by assuming each count comes from a Poisson distribution parameterized by latent factors for users and computers. The learned distributions are then used to predict unlikely authentication behavior.

Several variants of tiered recurrent networks have been explored in the machine learning and natural language processing communities (Koutnik et al. 2014; Ling et al. 2015b; 2015a; Chung et al. 2015). They are often realized by a lower tier pre-processing network, whose output is fed to an upper tier network and the separate tiers are jointly trained. Ling et al. (2015b) use a character-level convolutional neural network to feed a word level long short-term memory (LSTM) RNN for machine translation, with predictions made at the word-level. Both Hwang and Sung (2016) and Ling et al. (2015a) use a character-based LSTM to feed a second word or utterance-based LSTM for language modeling. Pascanu et al. (2015) create activity models from real world data on a per-event (command) basis and sequences of system calls are then modeled using RNN and echo state networks. The learned features are used to independently train neural network and logistic regression classifiers. Max pooling is applied to hidden layers of the unsupervised RNN for each time step in a session and the result is concatenated to the final hidden state to produce feature vectors for the classifier. This is similar to our tiered approach, in which we use the average of all hidden states concatenated with the final hidden state as input to the upper-tier RNN. In contrast, our model is completely unsupervised and all components are jointly trained.

3 Approach

Our approach learns *normal* behavior for users, processing a stream of computer and network log-lines as follows:

1. Initialize model weights randomly
2. For each day k in chronological order:
   (a) Given model M_{k-1}, produce log-line-level anomaly scores for all events in day k
   (b) Optionally, produce an aggregated anomaly score each user for day k (from the log-line-level scores)
   (c) Send per-user-day or per-user-event anomaly scores in rank order to analysts for inspection
   (d) Update model weights to minimize loss on all log-lines in day k, yielding model M_k

This methodology interleaves detection and training in an online fashion. In this section we detail the components of our approach.

3.1 Log-Line Tokenization

To work directly from arbitrary log formats, we treat log-lines as sequences of tokens. For this work, we consider two tokenization granularities: word-level and character-level.

For word tokenization, we assume that tokens in the log-line are delimited by a known character (e.g., space or comma). After splitting the log-lines on this delimiter, we define a shared vocabulary of "words" over all log fields, consisting of the sufficiently-frequent tokens appearing in the training set. To allow our model to handle previously unseen tokens, we add an "out of vocbulary" token to our vocabulary, <oov>. (For instance, not every IP address will be represented in a training set; likewise, new PCs and users are continually being added to large networks.) To ensure that <oov> has non-zero probability, we replace sufficiently infrequent tokens in the training data with <oov>. During evaluation, tokens not seen before are labeled <oov>. In order to accommodate shifting word distributions in an online environment, a fixed size vocabulary could be periodically

updated using a sliding window of word frequency statistics. For simplicity, we assume we have a fixed training set from which we produce a fixed vocabulary.

To avoid the challenges of managing a word-level vocabulary, we also develop language models using a character-level tokenization. In this case our primitive vocabulary, the alphabet of printable ASCII characters, circumvents the open vocabulary issue by its ability to represent any log entry irrespective of the network, logging source, or log field. With character-level tokenization, we keep the delimiter token in the sequence, to provide our models with cues to transitions between log-line fields.

3.2 Recurrent Neural Network Language Models

To produce log-line-level anomaly scores, we use recurrent neural networks in two ways: 1) as a language model over individual log-lines, and 2) to model the state of a user over time. We first present two recurrent models that focus only on (1), and then a tiered model that accomplishes both (1) and (2). Both were implemented[1] for our experiments using TensorFlow (Abadi et al. 2015).

Event Model (`EM`). First we consider a simple RNN model that operates on the token (e.g., word) sequences of individual log-lines (events). Specifically, we consider a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) network whose inputs are token embeddings and from whose output we predict distributions over the next token.

For a log-line with K tokens, each drawn from a shared vocabulary of size C, let $\mathcal{X}_{(1:K)} = \mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots, \mathbf{x}_{(K)}$ denote a sequence of one-hot representations of the tokens (each $\mathbf{x}_{(t)} \in \mathbb{R}^C$).

In this model, the hidden representation at token t, $\mathbf{h}_{(t)}$, from which we make our predictions, is a function of $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots, \mathbf{x}_{(t)}$ according to the usual LSTM equations:

$$\mathbf{h}_{(t)} = \mathbf{o}_{(t)} \circ \tanh(\mathbf{c}_{(t)}) \quad (1)$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \circ \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \circ \mathbf{g}_{(t)} \quad (2)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{x}_{(t)}\mathbf{W}_{(g,x)} + \mathbf{h}_{(t-1)}\mathbf{W}_{(g,h)} + \mathbf{b}_{(g)}\right) \quad (3)$$
$$\mathbf{f}_{(t)} = \sigma\left(\mathbf{x}_{(t)}\mathbf{W}_{(f,x)} + \mathbf{h}_{(t-1)}\mathbf{W}_{(f,h)} + \mathbf{b}_{(f)}\right) \quad (4)$$
$$\mathbf{i}_{(t)} = \sigma\left(\mathbf{x}_{(t)}\mathbf{W}_{(i,x)} + \mathbf{h}_{(t-1)}\mathbf{W}_{(i,h)} + \mathbf{b}_{(i)}\right) \quad (5)$$
$$\mathbf{o}_{(t)} = \sigma\left(\mathbf{x}_{(t)}\mathbf{W}_{(o,x)} + \mathbf{h}_{(t-1)}\mathbf{W}_{(o,h)} + \mathbf{b}_{(o)}\right), \quad (6)$$

where the initial hidden and cell states, $\mathbf{c}_{(0)}$ and $\mathbf{h}_{(0)}$, are set to zero vectors, and $\circ$ and σ denote element-wise multiplication and logistic sigmoid, respectively. Vector $\mathbf{g}_{(t)}$ is a hidden representation based on the current input and previous hidden state, while vectors $\mathbf{f}_{(t)}$, $\mathbf{i}_{(t)}$, and $\mathbf{o}_{(t)}$, are the standard LSTM gates. The matrices ($\mathbf{W}$) and bias vectors ($\mathbf{b}$) are the model parameters. We use each $\mathbf{h}_{(t-1)}$ to produce a probability distribution $\mathbf{p}_{(t)}$ over the token at time t, as follows:

$$\mathbf{p}_{(t)} = \text{softmax}\left(\mathbf{h}_{(t-1)}\mathbf{W}_{(p)} + \mathbf{b}_{(p)}\right) \quad (7)$$

[1]Code is available at https://github.com/pnnl/safekit

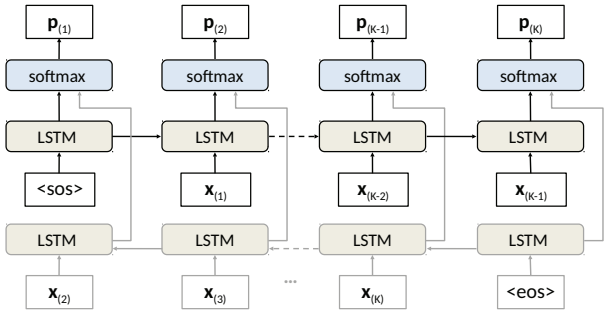

Figure 1: Event Models. Set of black bordered nodes and connections illustrate the `EM` model while set of all nodes and connections illustrate the `BEM` model.

We use cross-entropy loss,

$$\frac{1}{K}\sum_{t=1}^{K} H(\mathbf{x}_{(t)}, \mathbf{p}_{(t)}), \quad (8)$$

for two important purposes: first, as per-log-line anomaly score and second, as the training objective to update model weights. We train this model using stochastic mini-batch (non-truncated) back-propagation through time.

Bidirectional Event Model (`BEM`). Following the language model formulation suggested in (Schuster and Paliwal 1997), we alternatively model the structure of log lines with a *bidirectional* LSTM. We define a new set of hidden vectors $\mathbf{h}^b_{(K+1)}, \mathbf{h}^b_{(K)}, \ldots, \mathbf{h}^b_{(1)}$ by running the LSTM equations backwards in time (starting with initial zero cell and hidden states at time $K+1$ set to zero). The weights $\mathbf{W}$ and biases $\mathbf{b}$ for the backward LSTM are denoted with superscript b.

The probability distribution $\mathbf{p}_{(t)}$ over the token at time t is then:

$$\mathbf{p}_{(t)} = \text{softmax}\left(\mathbf{h}_{(t-1)}\mathbf{W}_{(p)} + \mathbf{h}^b_{(t+1)}\mathbf{W}^b_{(p)} + \mathbf{b}_{(p)}\right) \quad (9)$$

Tiered Event Models (`T-EM`, `T-BEM`). To incorporate inter-log-line context, we propose a two-tiered recurrent neural network. The lower-tier can be either event model (`EM` or `BEM`), but with the additional input of a context vector (generated by the upper-tier) concatenated to the token embedding at each time step. The input to the upper-tier model is the hidden states of the lower-tier model. This upper tier models the dynamics of user behavior over time, producing the context vectors provided to the lower-tier RNN. This model is illustrated in Fig. 2.

In this model, $x^{(u,j)}$ denotes user u's jth log line, which consists of a sequence of tokens as described in the previous subsections. The upper-tier models a sequence of user log lines, $x^{(u,1)}, x^{(u,2)}, \ldots, x^{(u,T_u)}$, using an LSTM. For each user u and each log line j in the user's log line sequence, a lower-tier LSTM is applied to the tokens of $x^{(u,j)}$. The input to the upper-tier model at log-line j is the concatenation of: 1) the final lower-tier hidden state(s) and 2) the average of the lower-tier hidden states. In the case of a lower-tier EM,

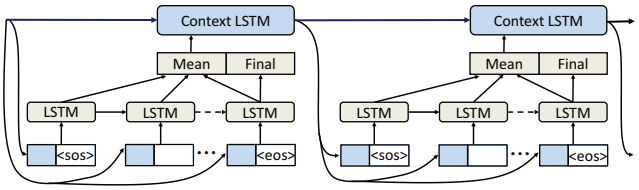

Figure 2: Tiered Event Model (T-EM)

(1) refers to the hidden state at time K; for the BEM, (1) is the concatenation of the forward hidden state at time K and the backward hidden state at time 1. For (2), we average over hidden states primarily to provide many short-cut connections in the LSTM, which aids trainability. The output of the upper-tier LSTM at log-line j is a hidden state $\hat{\mathbf{h}}^{(u,j)}$. This hidden vector serves to provide context for the lower-tier model at the next time step: specifically, $\hat{\mathbf{h}}^{(u,j-1)}$ is concatenated to each of the inputs of the lower-tier model operating on the jth log-line. Note that the upper-tier model serves only to propagate context information across individual log-lines; no loss is computed directly on the values produced by the upper-tier model.

The upper- and lower-tier models are trained jointly to minimize the cross-entropy loss of the lower-tier model. We unroll the two-tier model for a fixed number of log-lines, fully unrolling each of the lower-tier models within that window. The lower-tier model's cross-entropy loss is also used to detect anomalous behavior, as is described further in Section 4.2.

Minibatching becomes more challenging for the tiered model, as the number of log-lines per day can vary dramatically between users. This poses two problems: first, it introduces the possibility that the most active users may have a disproportionate impact on model weights; second, it means that toward the end of the day, there may not be enough users to fill the minibatch. To counteract the first problem, we fix the number of log-lines per user per day that the model will train on. The remaining log-lines are not used in any gradient updates. We leave compensating for the inefficiency that results from the second to future work.

3.3 Baselines

Anomaly detection in streaming network logs often relies upon computing statistics over windows of time and applying anomaly detection techniques to those vectors. Below we describe the aggregate features and two anomaly detection techniques that are typical of prior work.

Aggregate Features We first define the set of per-user-day features, which summarize users' activities in the day. To aggregate the features that have a small number of distinct values (e.g. success/failure, logon orientation) we count the number of occurrences for each distinct value for the user-day. For fields that have a larger number of distinct values (pcs, users, domains), we count the number of common and uncommon events that occurred, rather than the number of occurrences of each distinct value (this approach avoids high dimensional sparse features). Furthermore, we define two categories of common/uncommon; to the individual entity/user, and relative to all users. A value is defined as uncommon for the user if it accounts for fewer than 5% of the values observed in that field (up to that point in time), and common otherwise. A value is defined as uncommon for all users if it occurs fewer times than the average value for the field, and common otherwise.

For the LANL dataset, the prior featurization strategy yields a 108-dimensional aggregate feature vector per user-day. These feature vectors then serve as the input to the baseline models described next.

Models We consider two baseline models. The first uses Principal Components Analysis (`pca`) to learn a low dimensional representation of the aggregate features; the anomaly score is proportional to the reconstruction error after mapping the compressed representation back into the original dimension (Shyu et al. 2003). The second is an isolation forest (`iso`) based approach (Liu, Ting, and Zhou 2008) as implemented in scikit-learn's outlier detection tools (Pedregosa et al. 2011). This was noted as the best performing anomaly detection algorithm in the recent DARPA insider threat detection program, (Gavai et al. 2015).

4 Experiments

In this section we describe experiments to evaluate the effectiveness of the proposed event modeling algorithms.

4.1 Data

The Los Alamos National Laboratory (LANL) Cyber Security Dataset (Kent 2016) consists of event logs from LANL's internal computer network collected over a period of 58 consecutive days. The data set contains over one billion log-lines from authentication, process, network flow, and DNS logging sources. Identifying fields (e.g., users, computers, and processes) have been anonymized.

The recorded network activities included both normal operational network activity as well as a series of red team activities that compromised account credentials over the first 30 days of data. Information about known red team attack events is used only for evaluation; our approach is strictly unsupervised.

For the experiments presented in this paper, we rely only on the authentication event logs, whose fields and statistics are summarized in Figure 3a. We filter these events to only those log-lines linked to an actual user, removing computer-computer interaction events. Events on weekends and holidays contain drastically different frequencies and distributions of activities. In a real deployment a separate model would be trained for use on those days, but because no malicious events were in that data it was also withheld.

Table 3b has statistics of our data split; the first 12 days serve as the development set, while the remaining 18 days are the independent test set.

4.2 Assessment Granularity

Our model learns normal behavior and assigns relatively high loss to events that are unexpected. A principal advantage of our approach is this ability to score the anomaly of

Field	Example	# unique labels
time	1	5011198
source user	C625@DOM1	80553
dest. user	U147@DOM1	98563
source pc	C625	16230
dest. pc	C625	15895
auth. type	Negotiate	29
logon type	Batch	10
auth. orient	LogOn	7
success	Success	2

(a)

	Dev	Test
Days	1-12	13-58
# Events	$133M$	$918M$
# Attacks	316	385
# User-days	57	79

(b)

Figure 3: Dataset statistics: (a) Authentication log fields and statistics and (b) dataset splits.

individual events, allowing us to flag at the event-level or aggregate anomalies over any larger timescale.

For this work, we consider two timescales. First, we assess based on individual events; a list of events would be presented to the analyst, sorted descending by anomaly score. Second, to facilitate comparison with traditional aggregation methods, we aggregate anomaly scores over all of a user's events for the day (specifically, taking the max), producing a single anomaly score per-user, per-day. In this scenario, a list of user-days would be provided to the analyst, sorted descending by anomaly score. We refer to this approach as `max`, because the anomaly scores provided to the analyst are produced by taking the maximum score over the event scores in the window for that user (where event-level scoring is just taking the max over a singleton set of one event).

In order to counter systematic offsets in users' anomaly scores for a day we also consider a simple normalization strategy, which we refer to as `diff`, by which every raw score is first normalized by subtracting the user's average event-level anomaly score for the day.

4.3 Metrics

We consider two performance metrics. First, we assess results using the standard area under the receiver operator characteristic curve (AUC) which characterizes the trade-off in model detection performance between true positives and false positives, effectively sweeping through all possible analyst budgets. False positives are detections that are not truly red team events, while true positives are detections that are.

To quantify the proportion of the data the analyst must sift through to diagnose malicious behavior on the network, we use the Average Percentile (AP) metric. Specifically, for each red team event or user-day, we note the percentile of its anomaly amongst all anomaly scores for the day. We then average these percentiles for all of the malicious events or user-days. Note that if all true malicious events or user-days are flagged as the most anomalous on the respective days, then $\texttt{AP} \approx 100$, while if all malicious events or user-days are ranked as the least anomalous on their respective days, $\texttt{AP} \approx 0$. For both AUC and AP, a higher score is better.

Our model hyperparameters were manually tuned to maximize AP for day-level `diff` scores on the development set. No separate training set is needed as our approach is unsupervised and trained online.

Model	Tokenization	AUC	AP
`pca`	-	0.754	73.9
`iso`	-	0.763	75.0
`EM`	Word	0.802	79.3
`BEM`	Word	**0.876**	**87.0**
`T-EM`	Word	0.782	77.5
`T-BEM`	Word	0.864	85.7
`EM`	Char	0.750	70.9
`BEM`	Char	**0.843**	82.9
`T-EM`	Char	0.772	76.2
`T-BEM`	Char	0.837	**82.9**

Table 1: User-day granularity test set AUC and AP. Language model anomaly scores calculated with average user-day normalization (`diff`).

	LOG		DAY	
	diff	**max**	**diff**	**max**
`W EM`	0.964	0.932	0.802	0.794
`W BEM`	**0.974**	0.895	0.876	0.811
`W T-EM`	0.959	0.948	0.782	0.803
`W T-BEM`	0.959	0.902	0.864	0.838
`C EM`	0.940	0.935	0.751	0.754
`C BEM`	0.973	**0.979**	0.843	0.846
`C T-EM`	0.859	0.927	0.772	0.809
`C T-BEM`	0.945	0.969	0.837	0.854

Table 2: Comparison of AUC for day-level and log-line-level analysis with and without user-day normalization. Figures 4 and 5 provide a visualization of these results.

4.4 Results and Analysis

We begin by exploring the user-day granularity performance. Table 1 summarizes model detection performance at this granularity on the test set for the AUC and AP metrics using the `diff` method to produce day level scores from the language models. A few trends are evident from these results. First, the aggregate feature baselines have near-equivalent performance by both metrics, with the isolation forest approach having a slight edge. We hypothesize the feature representation, which is common to these methods, could be a bottleneck in performance. This highlights the "blind spot" issue feature engineering introduces. Second, despite having only the context of a single log-line at a time, as opposed to features aggregated over an entire day, the event model (`EM`) performs comparably to the baseline models when a forward pass LSTM network is used with

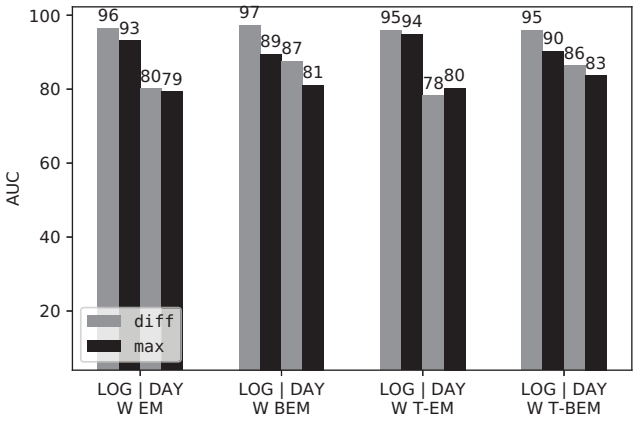

Figure 4: Word model comparison of AUC at day-level and log-line-level granularities.

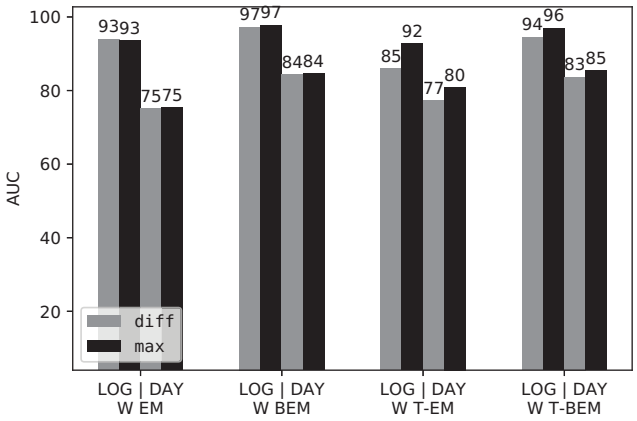

Figure 5: Character model comparison of AUC at day-level and log-line-level granularities.

a character tokenization and outperforms the baselines with word tokenization. The most pronounced performance gain results from using bidirectional models. Finally, word-level tokenization performs better than character-level; however, even the bidirectional *character* models perform appreciably better than the baselines.

It is clear from these results that the tiered models perform comparably to, but not better than, the event-level models. This suggests that the event level model is able to characterize normal user behavior from information stored in the model weights of the network, which are trained each day to model user activity. Given the context of the past day's activity stored in the model weights, the categorical variables represented by the fields in an individual log line may eliminate the need for explicit event context modeling. We leave tracking the state of individual computers, rather than users, to future work, but hypothesize that it may make the tiered approach more effective.

Next, we broaden our analysis of language modeling approaches, comparing performance across all language models, tokenization strategies, anomaly granularity, and normalization techniques. Figure 4 plots AUC for all language model types using word tokenization, contrasting `max` and `diff` normalization modes. Figure 5 compares the same variations for character tokenization. Table 2 presents these results in tabular form. With few exceptions, log-line-level granularity vastly outperforms day-level; this is true for both the character-level and word-level tokenization strategies, with an average gain of 0.1 AUC. The most interesting outcome of these comparisons is that word tokenization performance gains are heavily reliant on the `diff` normalization, whereas for character tokenization the `diff` normalization has a minor detrimental effect for some models. This suggests that the character-level model could be used to provide a more immediate response time, not having to wait until the day is done to obtain the day statistics used in `diff` mode. The two tokenization strategies may in fact be complementary as the versatility and response time gains of a character tokenization come at the expense of easy interpretibility of a word tokenization: the word tokenization allows anomaly scores to be decomposed into individual log-line fields, enabling analysts to pinpoint features of the event that most contributed to it being flagged. Since we tuned hyperparameters using `diff` mode, the character-level model has potential to do better with additional tuning.

Next, Figures 6 and 7 visualize the average percentiles of red team detections for the subset of the test set with the most red-team activity. Anomaly scores for both word and character tokenizations are computed *without* average user-day offset normalization. Red team log-line-level scores are plotted as purple x's with the x coordinate being the second in time at which the event occurred and y coordinate the anomaly score for that event. Percentile ranges are colored to provide context for the red-team anomaly scores against the backdrop of other network activity. The spread of non-normalized anomaly scores is much greater for the word-level tokenizations (Fig. 7) than character-level (Fig. 6), which could explain the different sensitivity of word level tokenization to normalization. Also notice that there is an expected bump in percentiles for windows of frequent red-team activity. Curiously, at the end of day 14 there are massive bumps for the 99th percentile, which suggest unplanned and un-annotated anomalous events on the LANL network for those hours. Notice that for the character tokenization almost all non-normalized red team anomaly scores are above the 95th percentile, with a large proportion above the 99th percentile.

Finally, Figure 8 plots the ROC curves for the best aggregate baseline (`iso`), the best user-day granularity language model (word `BEM`), and the best event-level granularity model (character `BEM`). It illustrates the qualitatively different curves obtained with the baselines, the user-day granularity, and the event-level granularity.

Since the proportion of red-team to normal events is vanishingly low in the data-set ($< 0.001\%$), the false-positive rate is effectively the proportion of data flagged to achieve a particular recall. From this observation, Figure 8 shows the character event model can achieve 100% recall from only 12% of the data whereas the other models considered only achieve 100% recall when nearly all of the data has been

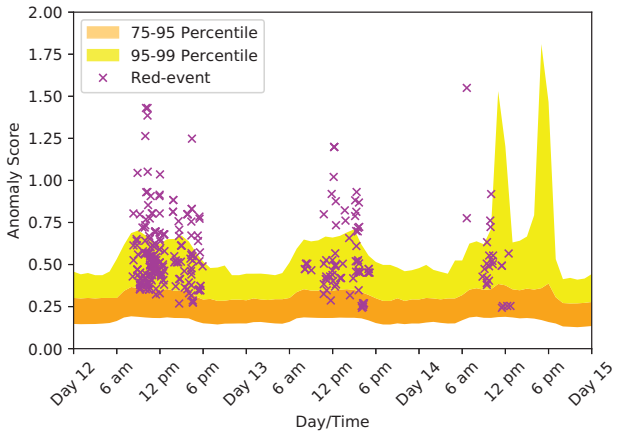

Figure 6: Character-level red-team log-line anomaly scores in relation to percentiles over time.

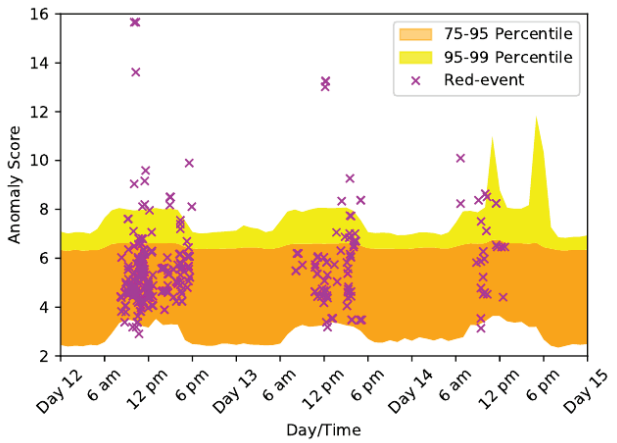

Figure 7: Word-level red-team log-line anomaly scores in relation to percentiles over time.

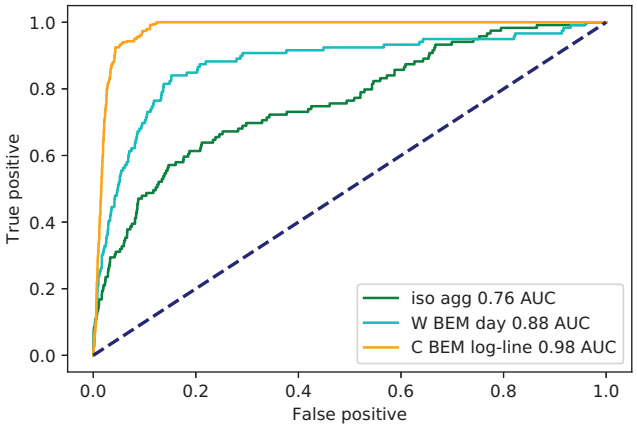

Figure 8: ROC curves for best performing baseline, word language model evaluated at day-granularity, and character language model evaluated at log-line-granularity.

handed to the analyst. Further, the character event model can achieve 80% recall by flagging only 3% of the data whereas the word day language model needs 14% of the data and the aggregate isolation forest model needs 55% of the data to achieve the same result.

5 Conclusion

This work builds upon advances in language modeling to address computer security log analysis, proposing an unsupervised, online anomaly detection approach. We eliminate the usual effort-intensive feature engineering stage, making our approach fast to deploy and agnostic to the system configuration and monitoring tools. It further confers the key advantage of event-level detection which allows for a near immediate alert response following anomalous activity.

In experiments using the Los Alamos National Laboratory Cyber Security Dataset, bidirectional language models significantly outperformed standard methods at day-level detection. The best log-line-level detection performance was achieved with a bidirectional character-based language model, obtaining a 0.98 area under the ROC curve, showing that for the constrained language domain of network logs, character based language modeling can achieve comparable accuracy to word based modeling for event level detection. We have therefore demonstrated a simple and effective approach to modeling dynamic networks with open vocabulary logs (e.g. with new users, PCs, or IP addresses).

We propose to extend this work in several ways. First, potential modeling advantages of tiered architectures merit further investigation. The use of tiered architectures to track PCs instead of network users, or from a richer set of logging sources other than simply authentication logs may take better advantage of their modeling power. Next, we anticipate interpretability can become lost with such detailed granularity provided by log-line-level detection from a character-based model, therefore future work will explore alternate methods of providing context to an analyst. Finally, we are interested in exploring the robustness of this approach to adversarial tampering. Similarly performing models could have different levels of resilience that would lead to selection of one over another.

Acknowledgments The research described in this paper is part of the Analysis in Motion Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy. The authors would also like to thank the Nvidia corporation for their donations of Titan X and Titan Xp GPUs used in this research.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah,

C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Alrawashdeh, K., and Purdy, C. 2016. Toward an online anomaly intrusion detection system based on deep learning. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, 195–200. IEEE.

Bhattacharyya, D. K., and Kalita, J. K. 2013. *Network anomaly detection: A machine learning perspective*. CRC Press.

Bivens, A.; Palagiri, C.; Smith, R.; Szymanski, B.; Embrechts, M.; et al. 2002. Network-based intrusion detection using neural networks. *Intelligent Engineering Systems through Artificial Neural Networks* 12(1):579–584.

Buczak, A. L., and 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.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2015. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, 2067–2075.

Debar, H.; Becker, M.; and Siboni, D. 1992. A neural network component for an intrusion detection system. In *Proc. IEEE Symposium on Research in Security and Privacy*, 240–250.

Dua, S., and Du, X. 2016. *Data mining and machine learning in cybersecurity*. CRC press.

Gavai, G.; Sricharan, K.; Gunning, D.; Hanley, J.; Singhal, M.; and Rolleston, R. 2015. Supervised and unsupervised methods to detect insider threat from enterprise social and online activity data. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 6(4):47–63.

Gopalan, P.; Hofman, J. M.; and Blei, D. M. 2013. Scalable recommendation with Poisson factorization. *arXiv preprint arXiv:1311.1704*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hwang, K., and Sung, W. 2016. Character-level language modeling with hierarchical recurrent neural networks. *arXiv preprint arXiv:1609.03777*.

Kent, A. D. 2016. Cyber security data sources for dynamic network research. *Dynamic Networks and Cyber-Security* 1:37.

Koutnik, J.; Greff, K.; Gomez, F.; and Schmidhuber, J. 2014. A clockwork RNN. *arXiv preprint arXiv:1402.3511*.

Kumar, G.; Kumar, K.; and Sachdeva, M. 2010. The use of artificial intelligence based techniques for intrusion detection: a review. *Artificial Intelligence Review* 34(4):369–387.

Ling, W.; Luís, T.; Marujo, L.; Astudillo, R. F.; Amir, S.; Dyer, C.; Black, A. W.; and Trancoso, I. 2015a. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

Ling, W.; Trancoso, I.; Dyer, C.; and Black, A. W. 2015b. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*.

Liu, F. T.; Ting, K. M.; and Zhou, Z.-H. 2008. Isolation forest. In *Proc. ICDM*.

Novakov, S.; Lung, C.-H.; Lambadaris, I.; and Seddigh, N. 2013. Studies in applying PCA and wavelet algorithms for network traffic anomaly detection. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, 185–190. IEEE.

Pascanu, R.; Stokes, J. W.; Sanossian, H.; Marinescu, M.; and Thomas, A. 2015. Malware classification with recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 1916–1920. IEEE.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Ringberg, H.; Soule, A.; Rexford, J.; and Diot, C. 2007. Sensitivity of PCA for traffic anomaly detection. In *SIGMETRICS*.

Rubin-Delanchy, P.; Lawson, D. J.; and Heard, N. A. 2016. Anomaly detection for cyber security applications. *Dynamic Networks and Cyber-Security* 1:137.

Schuster, M., and Paliwal, K. K. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681.

Shyu, M.-L.; Chen, S.-C.; Sarinnapakorn, K.; and Chang, L. 2003. A novel anomaly detection scheme based on principal component classifier. In *Proc. ICDM*.

Sommer, R., and Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. Symposium on Security and Privacy*.

Tuor, A.; Kaplan, S.; Hutchinson, B.; Nichols, N.; and Robinson, S. 2017. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In *Artificial Intelligence for Cybersecurity Workshop at AAAI*.

Turcotte, M.; Moore, J.; Heard, N.; and McPhall, A. 2016. Poisson factorization for peer-based anomaly detection. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*, 208–210. IEEE.

Turcotte, M. J.; Heard, N. A.; and Kent, A. D. 2016. Modelling user behavior in a network using computer event logs. *Dynamic Networks and Cyber-Security* 1:67.

Veeramachaneni, K.; Arnaldo, I.; Korrapati, V.; Bassias, C.; and Li, K. 2016. AI^2: Training a big data machine to defend. In *Proc. HPSC and IDS*.

Zuech, R.; Khoshgoftaar, T. M.; and Wald, R. 2015. Intrusion detection and big heterogeneous data: a survey. *Journal of Big Data* 2(1):3.