Recurrent Neural Network Attention Mechanisms for Interpretable System Log Anomaly Detection

Andy Brown*
Western Washington University
browna52@wwu.edu

Aaron Tuor*
Pacific Northwest National Laboratory
aaron.tuor@pnnl.gov

Brian Hutchinson†
Western Washington University
brian.hutchinson@wwu.edu

Nicole Nichols
Pacific Northwest National Laboratory
nicole.nichols@pnnl.gov

ABSTRACT

Deep learning has recently demonstrated state-of-the art performance on key tasks related to the maintenance of computer systems, such as intrusion detection, denial of service attack detection, hardware and software system failures, and malware detection. In these contexts, model interpretability is vital for administrator and analyst to trust and act on the automated analysis of machine learning models. Deep learning methods have been criticized as black box oracles which allow limited insight into decision factors. In this work we seek to bridge the gap between the impressive performance of deep learning models and the need for interpretable model introspection. To this end we present recurrent neural network (RNN) language models augmented with attention for anomaly detection in system logs. Our methods are generally applicable to any computer system and logging source. By incorporating attention variants into our RNN language models we create opportunities for model introspection and analysis without sacrificing state-of-the art performance. We demonstrate model performance and illustrate model interpretability on an intrusion detection task using the Los Alamos National Laboratory (LANL) cyber security dataset, reporting upward of 0.99 area under the receiver operator characteristic curve despite being trained only on a single day's worth of data.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; *Online learning settings*; *Feature selection*; Unsupervised learning; Neural networks; Machine learning algorithms;

KEYWORDS

Anomaly detection, Attention, Recurrent Neural Networks, Interpretable Machine Learning, Online Training, System Log Analysis

*First and second authors contributed equally to this work.
†Joint appointment at PNNL and WWU (brian.hutchinson@pnnl.gov).

MLCS'18, June 12, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5865-1/18/06…$15.00
https://doi.org/10.1145/3217871.3217872

1 INTRODUCTION

System log analysis is critical for a wide range of tasks in maintaining large scale computer systems such as enterprise computer networks and high performance computing clusters. These include security tasks such as intrusion detection, insider threat detection, and malware detection, as well as more general maintenance tasks such as detecting hardware failure and modeling data or traffic flow patterns. Extracting knowledge from information rich system logs is complicated by several factors:

(1) Log sources can generate terabytes of data per day.
(2) Labeled data for application areas of interest is often scarce, unbalanced, or system specific.
(3) Actionable information may be obscured by complex, undiscovered relationships across logging sources and system entities (e.g. users, PCs, processes, nodes).

Due to these factors, unaided human monitoring and assessment is impractical, so considerable research has been directed to automated methods for visualization and analysis of system logs. Furthermore, as administrative decisions may be of considerable consequence to organizations and associated persons, it is crucial to have some understanding of the factors involved in automated decision processes, even for highly effective algorithms.

Addressing these factors, we present unsupervised recurrent neural network (RNN) language models for system log anomaly detection. By modeling the normal distribution of events in system logs, the anomaly detection approach can discover complex relationships buried in these logs. Since the methods are unsupervised, the models do not depend on the time consuming and otherwise expensive procurement of labeled data. Our language modeling framework requires little to no feature engineering: it is applicable to any serializable logging source. Further, the models are trained online using bounded resources dictated by the daily volume of the log sources. unsupervised learning, online training

Our main contributions in this work are twofold: 1) we evaluate the effectiveness of augmenting RNN language models with several attention mechanisms specifically designed for system log anomaly detection, and 2) we illustrate how model introspection in these systems is made possible by the attention mechanisms.

2 RELATED WORK

Recently, several researchers have used Long Short-Term Memory (LSTM) Networks [7] in system log analysis. Zhang et al. [23] use clustering techniques on the raw text from multiple log sources

minimal pre-processing

to generate feature sequences fed to an LSTM for hardware and software failure predictions. Du et al. [5] employ customized parsing methods on the raw text of system logs to generate sequences for LSTM Denial of Service attack detection. In contrast to these methods our approach works directly with raw text with no pre-processing beyond tokenization using known field delimiters. Others have incorporated LSTM networks to preprocess sequences of process API calls as components to malware detection systems [14] trained on labeled malware examples.

Attention-equipped LSTM models have been used to improve performance on complex sequence modeling tasks. Attention provides a dynamic weighted average of values from different points in a calculation during the processing of a sequence to provide long term context for downstream discriminative or generative prediction. In recent work [4, 22], researchers have augmented LSTM language models with attention mechanisms in order to add capacity for modeling long term syntactic dependencies. Yogatama et al. [22] characterize attention as a differentiable random access memory. They compare attention language models with differentiable stack based memory [6] (which provides a bias for hierarchical structure), demonstrating the superiority of stack based memory on a verb agreement task with multiple attractors. Daniluk et al. [4] explore three additive attention [2] mechanisms with successive partitioning of the output of the LSTM; splitting the output into separate key, value, and prediction vectors performed best, likely due to removing the need for a single vector to encode information for multiple steps in the computation. In contrast we augment our language models with dot product attention [11, 18], but also use separate vectors for the components of our attention mechanisms.

Many decision processes raise ethical dilemmas [12] or are applied in critical domains with high consequence. Such factors necessitate human interpretation of how a model is generating its predictions to ensure acceptable results. Vellido et al. [19] observe the gap between data modeling, knowledge extraction, and potential machine learning solutions, underscoring the need for interpretable automated decision processes. However, interpretability has multiple goals that are not always aligned with production of the most generalizable model architecture [10]. Hence, there is currently a large research focus on making interpretable deep learning algorithms for sensitive and critical application areas. Some proposed model introspection techniques include dimensionality reduction [20], analysis of intermediate layers [1] and saliency based methods [3, 13]. In contrast to other deep learning components, attention mechanisms allow an immediate view into what factors are affecting model decisions. Xu et al. [21] examine attention weights to determine what convolutional neural networks are "looking" at while making predictions. Similarly, Rocktäschel et al. [15] analyze matrices of word-to-word attention weights for insight into how their LSTM entailment classifier reasons about sentences. We apply the same concept to explore what factors our models attend over when predicting anomaly scores.

3 METHODS

Here we describe the unsupervised language modeling framework and its extension via five variations of attention. In each case, the language models consume a sequence of log-line tokens and output log-line-level anomaly scores.

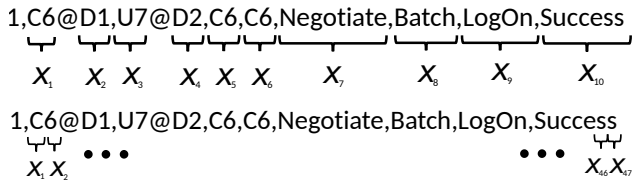

Figure 1: Top: Word tokens; Bottom: Character tokens

3.1 Preliminaries

3.1.1 Language Modeling. We assume that each log-line consists of a sequence of T tokens: $x_{(1:T)} = x_{(1)}, x_{(2)}, \ldots, x_{(T)}$. Each token $x_{(t)} \in \mathbb{V}$, where $\mathbb{V}$ denotes the vocabulary. A *language model* is a model that assigns probabilities to sequences: $P(x_{(1:T)})$. A language model often evaluates the probability of a sequence using the chain rule of probability:

$$P(x_{(1:T)}) = \prod_{t=1}^{T} P(x_{(t)} | x_{(<t)}) \tag{1}$$

where $x_{(<t)}$ denotes the (potentially empty) sequence of tokens from $x_{(1)}$ to $x_{(t-1)}$. The conditional probabilities on the righthand side can be modeled with a recurrent neural network, as will be described in the Section 3.2.

Our data consist of a series of log-lines, each affiliated with a user. We denote user u's ith log-line with $x^{(u,i)}_{(1:T)}$, but omit the superscript when it is non-essential. Our language models all output a single anomaly score, the negative log-likelihood, for each log-line.

3.1.2 Tokenization. Figure 1 illustrates two methods to partition log lines into sequences of tokens: word and character tokenization. For word based language modeling, the tokens are the fields of the CSV format log file. The user fields are further split on the "@" character to generate user name and domain tokens. A frequency threshold is applied to replace infrequent words with an "out of vocabulary" (OOV) token; a value must occur in a field at least 40 times to be added to the vocabulary. The OOV token ensures that our models will have non-zero probabilities when encountering previously unseen words during evaluation.

For character based language modeling we use a primitive vocabulary consisting of printable ASCII characters. This circumvents the out of vocabulary issues with the word model. Delimiters are left in the character inputs to give context of switching fields to the models. For both word and character tokenization, the time field is ignored and not tokenized.

3.2 Cyber Anomaly Language Models

We recently introduced a language modeling framework for cyber anomaly [17] that forms the starting point of this work. The first of four models presented in [17] is the "Event Model" (EM), which applies a standard LSTM [7] to the token sequences of individual events (log-lines). In order to feed the categorical tokens $x_{(1:T)}$ into the model, we first perform an embedding lookup on each token

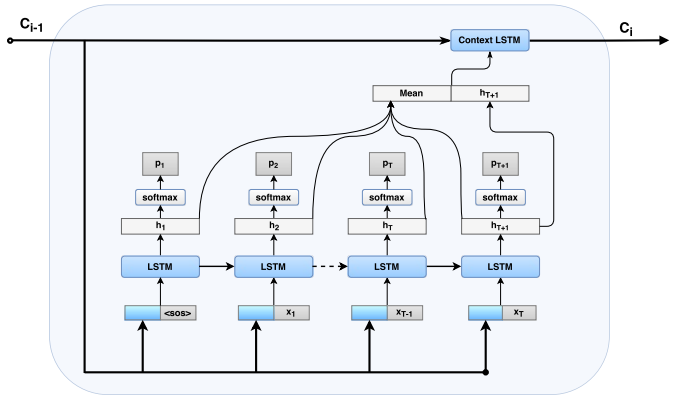

Figure 2: Tiered language model (T-EM) [17].

to yield the sequence $\mathbf{x}_{(1:T)}$ (bold font), where each $\mathbf{x}_{(t)} \in \mathbb{R}^{L_{emb}}$ for some embedding dimension hyperparameter, L_{emb}. There are unique embedding vectors for each element in the vocabulary; these embedding vectors are parameters of the model, learned jointly with all other model parameters. An LSTM maps an embedding vector sequence to a sequence of hidden vectors $\mathbf{h}_{(1:T)}$:[1]

$$\text{LSTM}(\mathbf{x}_{(1:T)}) = \mathbf{h}_{(1:T)} \tag{2}$$

Intuitively, $\mathbf{h}_{(t)}$ is a summary of the input sequence $\mathbf{x}_{(1:t)}$ defined by the same, standard LSTM equations used in [17]. Given the previous hidden state $\mathbf{h}_{(t-1)}$, weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$, the probability distribution over the token at step t is:

$$\mathbf{p}_{(t)} = \text{softmax}\Big(\underbrace{\mathbf{h}_{(t-1)}}_{L_h}\ \underbrace{\mathbf{W}}_{L_h \times |\mathbb{V}|} + \underbrace{\mathbf{b}}_{|\mathbb{V}|}\Big) \in \mathbb{R}^{|\mathbb{V}|} \tag{3}$$

This conditions each prediction on all tokens that precede it in the log-line. The second model in [17] is the Bidirectional Event Model (BEM), which updates Eqn. 3 to also incorporate the hidden state from a backward-running LSTM, with hidden vector $\mathbf{h}^b_{(t+1)}$ and additional weight matrix $\mathbf{W}^b$ as follows:

$$\mathbf{p}_{(t)} = \text{softmax}\left(\mathbf{h}_{(t-1)}\mathbf{W} + \mathbf{h}^b_{(t+1)}\mathbf{W}^b + \mathbf{b}\right) \tag{4}$$

The BEM conditions each prediction on the all of the other tokens in the log-line (preceding or following), for richer context.

The EM and BEM only condition predictions on other tokens in the same log line. However, Tuor et al. [17] also introduce tiered language model variants that employ an "upper tier" LSTM to model a user's sequence of log-lines (see Fig. 2). Each log-line is still modeled by an EM or BEM, but the input is the concatenation of embedding vectors $\mathbf{x}_t$ along with a *context vector* produced by the upper tier LSTM. The upper tier LSTM takes as input a summary of the lower-tier hidden states (the average lower-tier hidden state concatenated with the final hidden state). The upper and lower tiers are trained jointly. For later reference, we name these models T-EM and T-BEM, respectively.

[1]In this paper we assume all vectors are row vectors and adopt the notation convention of left multiplying matrices with row vectors (omitting the conventional transpose to avoid clutter).

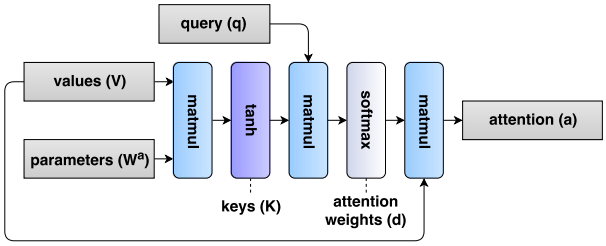

Figure 3: Dot Product Attention.

For all language models (including the tiered models which incorporate inter-log-line context) we optimize the model parameters by minimizing the negative log-likelihood produced by EM or BEM predictions. The negative log-likelihood minimization objective also serves as the anomaly score for the log line (less probable events receiving higher anomaly scores).

3.3 Attention

In this work we use dot product attention (Figure 3), wherein an "attention vector" $\mathbf{a}$ is generated from three values: 1) a key matrix $\mathbf{K}$, 2) a value matrix $\mathbf{V}$, and 3) a query vector $\mathbf{q}$. In this formulation, keys are a function of the value matrix:

$$\mathbf{K} = \tanh(\mathbf{V}\mathbf{W}^a), \tag{5}$$

parameterized by $\mathbf{W}^a$. The importance of each timestep is determined by the magnitude of the dot product of each key vector with the query vector $\mathbf{q} \in \mathbb{R}^{L_a}$ for some attention dimension hyperparameter, L_a. These magnitudes determine the weights, $\mathbf{d}$ on the weighted sum of value vectors, $\mathbf{a}$:

$$\mathbf{d} = \text{softmax}(\mathbf{q}\mathbf{K}^T) \tag{6}$$

$$\mathbf{a} = \mathbf{d}\mathbf{V} \tag{7}$$

In an LSTM, the information relevant to a given prediction (e.g. of the next token, $x_{(t+1)}$) is accumulated and propagated via the LSTM's cell state, $\mathbf{c}_{(t)}$. For any given prediction, however, certain tokens are likely to be more relevant than others. Attention provides a mechanism for predictions to be directly, selectively conditioned on a subset of the relevant tokens. In practice, this is accomplished by making $\mathbf{p}_{(t)}$ a function of the concatenation of $\mathbf{h}_{(t-1)}$ with an attention vector $\mathbf{a}_{(t-1)}$ that is a weighted sum over hidden states :

$$\mathbf{p}_{(t)} = \text{softmax}\left(\begin{bmatrix}\mathbf{h}_{(t-1)} & \mathbf{a}_{(t-1)}\end{bmatrix}\mathbf{W} + \mathbf{b}\right) \tag{8}$$

This attention mechanism not only introduces shortcuts in the flow of information over time, allowing the model to more readily access the relevant information for any given prediction, but the weights on the weighted sum also yield insights into the model's decision process, aiding interpretability.

We first examine the case of adding attention to the standard EM. Each token-step t is associated with its own value matrix $\mathbf{V}_{(t)}$, and query vector $\mathbf{q}_{(t)}$. The value matrix $\mathbf{V}_{(t)}$ is the matrix of hidden states up to but excluding token-step t, where L_h is the dimension of the LSTM hidden states. These are the values over which the weighted sum will be performed.

$$\mathbf{V}_{(t)} = \begin{bmatrix} \mathbf{h}_{(1)} \\ \vdots \\ \mathbf{h}_{(t-1)} \end{bmatrix} \in \mathbb{R}^{(t-1)\times L_h} \tag{9}$$

From the value matrix and weight matrix $\mathbf{W}^a \in \mathbb{R}^{L_h \times L_a}$, we compute a set of keys for each token/step:

$$\mathbf{K}_{(t)} = \tanh(\mathbf{V}_{(t)}\mathbf{W}^a) \in \mathbb{R}^{(t-1)\times L_a} \tag{10}$$

Then,

$$\mathbf{d}_{(t)} = \text{softmax}(\mathbf{q}_{(t)}\mathbf{K}^T_{(t)}) \tag{11}$$

$$\mathbf{a}_{(t)} = \mathbf{d}_{(t)}\mathbf{V}_{(t)} \tag{12}$$

Our EM attention variants differ primarily in the definition of the query vector $\mathbf{q}_{(t)}$.

3.3.1 Fixed Attention. In the fixed variation of attention [16] we let $\mathbf{q}_{(t)} = \mathbf{q}$ for some fixed, learned vector $\mathbf{q}$ that is shared across all tokens/steps. This assumes some positions in the sequence are more important than others, but that importance does not depend on the token one is trying to predict.

3.3.2 Syntax Attention. Syntax attention differs from fixed attention in that $\mathbf{q}_{(t)}$ is not shared across t. This assumes that some tokens are more important than others and this importance depends on the position in the sequence for the token to predict, but not on the actual values for tokens $x_1, \ldots, x_{t-1}$.

3.3.3 Semantic Attention. For semantic attention the query is a function of the current input token, and so the resulting attention weights are conditoned on the relations between content from different positions in the sequence. Such modeling is necessary for sequences which do not have a fixed role assigned to each position, as in the case of free form logging formats, or character based tokenization.

Specifically, for the first "semantic" variation, our query is a a function of the current hidden state and parameter matrix $\mathbf{W}^{sem1} \in \mathbb{R}^{L_h \times L_a}$:

$$\mathbf{q}_{(t)} = \tanh(\mathbf{h}_{(t)}\mathbf{W}^{sem1}) \tag{13}$$

In our other variant, Semantic Attention II, instead of making $\mathbf{q}_{(t)}$ a function of $\mathbf{h}_{(t)}$, we interpret each $\mathbf{h}_{(t)}$ emitted from the LSTM as the concatenation of two vectors: $\mathbf{h}'_{(t)}$ and $\mathbf{q}_{(t)}$. The query portion, $\mathbf{q}_{(t)}$ is used as before, but now the value $\mathbf{V}_{(t)}$ defined in Eqn. 9 contains $\mathbf{h}'_{(1)}$ through $\mathbf{h}'_{(t-1)}$. Note that, per the LSTM equations, both $\mathbf{h}'_{(t)}$ and $\mathbf{q}_{(t)}$ will be fed back into the LSTM at time $t+1$.

3.3.4 Tiered Attention. As shown in Fig. 2, in original formulation of the tiered model, the lower tier LSTM hidden states are averaged in the process of passing information from the lower tier to the upper tier. Implementation of attention for the tiered language models replaces this mean with a weighted average via attention. Let $\mathbf{V}^{(u,i)}$ be the lower tier hidden states for user u's ith log line:

$$\mathbf{V}^{(u,i)} = \begin{bmatrix} \mathbf{h}^{(u,i)}_{(1)} \\ \vdots \\ \mathbf{h}^{(u,i)}_{(T)} \end{bmatrix} \tag{14}$$

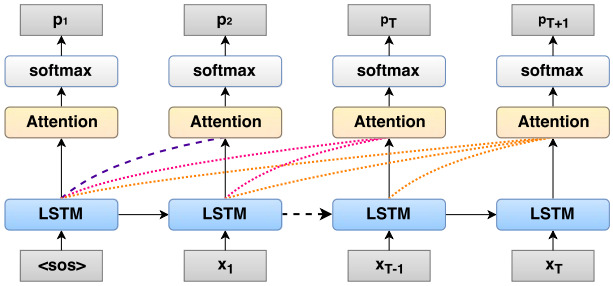

Figure 4: Event Model (EM) with attention. Dotted lines indicate which hidden states are being attended over.

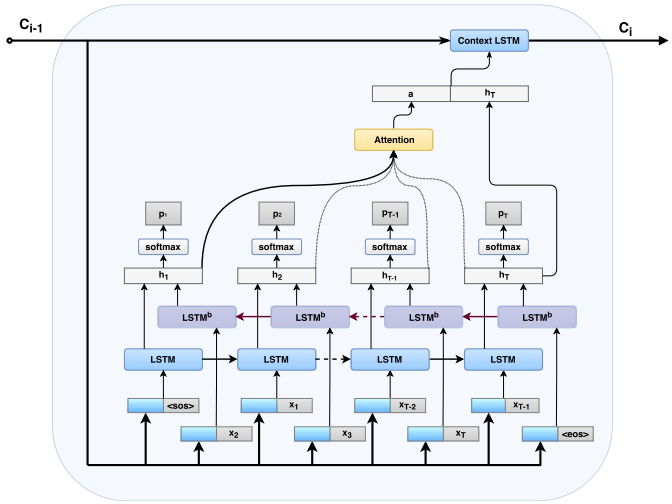

Figure 5: Tiered attention with bidirectional lower tier

Let $\mathbf{W}^{tier} \in \mathbb{R}^{L_k \times L_a}$ and $\mathbf{W}^a \in \mathbb{R}^{L_a \times L_k}$ be parameter matrices. We then define the following attention mechanisms:

$$\mathbf{q} = \tanh(\mathbf{h}_{(T)}\mathbf{W}^{(tier)}) \tag{15}$$

$$\mathbf{K} = \tanh(\mathbf{V}\mathbf{W}^a) \tag{16}$$

$$\mathbf{d} = \text{softmax}(\mathbf{q}\mathbf{K}^T) \tag{17}$$

$$\mathbf{a} = \mathbf{d}\mathbf{V} \tag{18}$$

We then replace the average of the hidden states in the tiered model with $\mathbf{a}$. Note that each sequence shares weights $\mathbf{W}^a$ and $\mathbf{W}^{tier}$. The BEM tiered attention model (TA-BEM) is depicted in Figure 5.

3.4 Online Training

We employ a simple online algorithm for training and prediction, which both allows our model to continually adapt to changing distributions of activities on a network and to be deployed on high throughput streaming data sources. At the beginning of each day/cycle, the parameters of the current model are fixed for evaluation, thereby avoiding evolving anomaly score scale issues that could result from continuous online training. After anomaly scores have been calculated for the day's events we train on the current day's events. The days events are then discarded, bounding the storage demands of the algorithm to a day's worth of activity (plus

model parameters). On the first day we do not evaluate as the model has not had a training phase yet.

At the cost of the additional space complexity of storing two copies of the model parameters, the training and evaluation phases can be run concurrently. The evaluation and training parameters are then synced daily so that the evaluation copy is updated with the parameters of the training copy at the beginning of each day.

4 EXPERIMENTS

This section discusses the data, experimental setup, evaluation metrics, and results assessing performance for the proposed methods.

4.1 Data

We evaluate our models on the publicly available LANL [8] dataset. LANL consists of over one billion log lines collected over 58 consecutive days. The logs contain anonymized process, network flow, DNS, and authentication information. Interleaved are attacks by a red team. Our experiments focus on modeling the authentication logs, which contain the following fields:

> Source user, Destination user, Source pc, Destination pc, Authentication type, Logon type, Authentication orientation, Success/failure.

These events are collected from desktop PCs, servers, and active directory servers using the Windows OS. We filter automated system events by discarding all log-lines that have a machine listed as the source user. Red team event log-lines are indicated in the dataset. As our models are fully unsupervised, we use the red team labels only for evaluation of model performance.

4.2 Experimental Setup

To assess our model's ability to spin up rapidly and detect anomalies with minimal burn-in time, we limit our scope to days 7 and 8, which contain 1 and 261 red team events respectively. Each of these days contains over seven million user log lines. We chose these particular days for evaluation because day 8 has the largest number of red events in the dataset. The entire experimental process is therefore 1) train on day 7, 2) evaluate on day 8. Further simulating a rapid deployment process, we performed no hyper-parameter tuning. Our learning rate is fixed to 0.01; we train using the ADAM [9] optimizer; the minibatch size is 64; our LSTMs have a single layer with 128 hidden units; our token embedding size is 128 and our attention size is 128. To estimate model variability, we trained each model five times with the fixed hyper-parameters but different random weight initializations. In our results section we report statistics over the five runs.

4.3 Metrics and Score Normalization

We evaluate our results using the area under the receiver operating characteristic curve (AUC ROC). ROC plots the true positive rate against the false positive rate as the detection threshold is swept. Perfect detection yields an AUC of 1 and random guessing yields 0.5. Recall that our anomaly scores, $z^{(u,i)}$ are given by the sum of the negative log probabilities of the tokens in line $\mathbf{x}^{(u,i)}_{(1:T)}$. For word tokenization, we center each user's anomaly score:

$$z^{(u,i)} \leftarrow z^{(u,i)} - \frac{1}{N_u}\sum_i z^{(u,i)}, \forall_u, \tag{19}$$

where N_u is the number of events by user u in the day. This reduces inter-user anomaly bias, which can stem from the uneven distribution of user name tokens. This normalization is unnecessary for the character tokenization, as the user names are composed from a common character vocabulary.

Model	Mean	Max	Min	Std. Dev.
EM	0.968	0.976	0.964	0.005
BEM	0.976	0.981	0.972	0.003
EM with attention				
Fixed	0.974	0.976	0.972	**0.001**
Syntactic	0.972	0.975	0.967	0.004
Semantic 1	0.975	0.980	0.971	0.004
Semantic 2	0.973	0.976	0.968	0.003
Tiered LSTM variants				
T-EM	0.984	0.989	0.977	0.005
T-BEM	0.987	0.989	**0.985**	0.002
TA-EM	0.985	**0.991**	0.979	0.004
TA-BEM	**0.988**	**0.991**	0.984	0.003

Table 1: AUC statistics for word tokenization models

Model	Mean	Max	Min	Std. Dev.
EM	0.965	0.969	0.961	0.003
BEM	0.985	0.987	0.982	0.002
EM with attention				
Fixed	0.963	0.971	0.937	0.015
Syntactic	0.967	0.973	0.963	0.004
Semantic	0.975	0.977	0.971	0.003
Semantic 2	0.972	0.977	0.967	0.004
Tiered LSTM variants				
T-EM	0.977	0.988	0.967	0.008
T-BEM	**0.992**	**0.992**	**0.991**	**0.000**
TA-EM	0.982	0.984	0.979	0.002
TA-BEM	0.991	**0.992**	0.990	0.001

Table 2: AUC statistics for character tokenization models

4.4 Results

In this section we discuss performance of the different attention mechanisms. We note that variance of model performance across random parameter initializations is quite low for most models. This low variance given only a single day of pretraining suggests our method behaves predictably despite rapid deployment.

4.4.1 Word Tokenization Models. Table 1 shows AUC statistics for the word tokenization model experiments. Comparing the

word level LSTM baselines, the BEM outperforms the EM. However, adding attention to the EM improves performance to match the BEM. All variations of attention have very similar AUC scores. We hypothesize that the word model equally benefits from Syntax and Semantic attention, as it has a consistent syntax structure. Tiered word models with attention do not demonstrate as significant performance gains, however, both forward and bidirectional attention models trend slightly upwards in mean and maximum values from their non-attention counterparts.

4.4.2 Character Tokenization Models. As shown in Table 2, the Fixed and Syntax attention models appear ill-suited for character-based models with variable length fields; neither Fixed nor Syntax attention improve performance here, and the character EM model augmented with Fixed attention has a standard deviation 2-15 times that of other models. In contrast, semantic variants, where the attention weights are a function of the current input as opposed to sequence position, do improve performance but are not on par with the BEM. For the tiered models, we see little difference by incorporating attention, suggesting the shortcuts introduced by attention are unnecessary to propagate user context across log-lines. One interesting outcome is that a tiered model with either attention or a bidirectional lower tier has reduced variance across random initializations by a large factor for the character models.

5 ANALYSIS

While attention performs comparably to bidirectionality, it offers substantial advantages in its interpretability. Investigating which fields the model is attending to (and when) offers clues to its decision-making. In this section we illustrate two approaches to analysis of attention-equipped LSTM language models: 1) Analysis of global model behavior from summary statistics of attention weights, and 2) analysis of particular model decisions from case studies of attention weights and language model predictions.

5.1 Global Behavior

We can gain insight into the global behavior of an attention-equipped LSTM from summary statistics such as the mean and standard deviation of attention weights over the course of a day's predictions. Figure 6 shows the average attention weights for each EM attention model when predicting the last meaningful token (Success/Fail). Error bars of one standard deviation are shown to illustrate the variability of these weights.

Heatmaps of average attention weights for the four EM attention models proposed in Section 3.3 are provided in Figures 7, 8, 9 and 10. Each time step in a sequence generates its own set of weights, $\mathbf{d}_{(t)}$, over the previous hidden states. The larger the weight values are the more relevant the associated hidden state is to the current prediction. Note that the first input token is excluded from our figures as it has no previous hidden states to attend over.

5.1.1 Fixed. Figure 7 shows the mean weights for the Fixed attention which has a single fixed query that does not change with the context at the current time step. The source user, destination domain and source PC dominate the weight vectors, suggesting that they are the most important fields to this model.

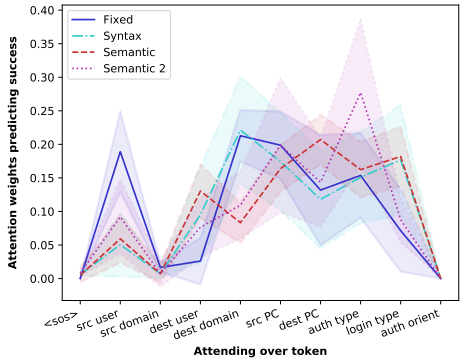

Figure 6: Comparison of attention weights when predicting success/failure token.

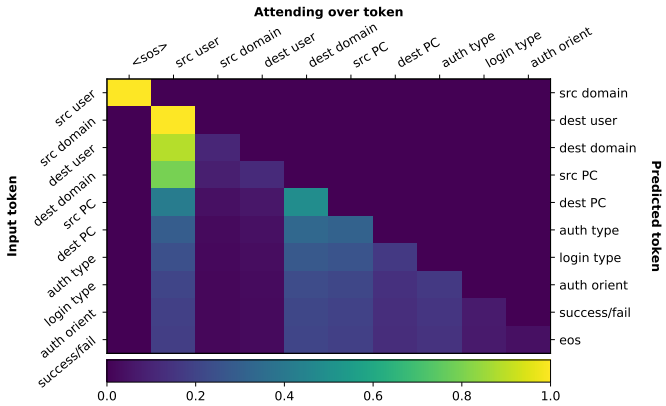

Figure 7: Average Fixed attention weights. The left y-axis is the current token seen by the RNN, while the right y-axis is the token being predicted. The x-axis shows the attention weight for for each previous token.

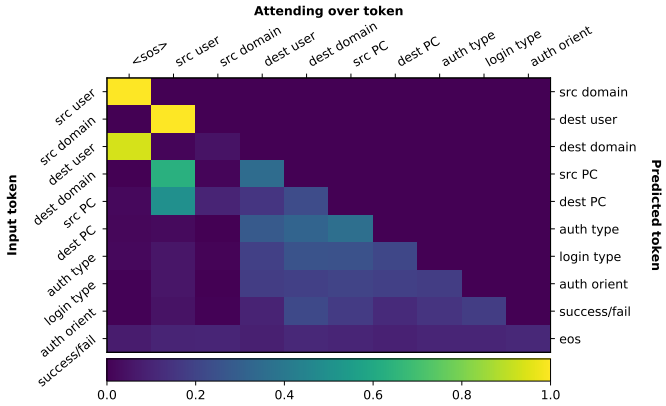

Figure 8: Average Syntax attention weights.

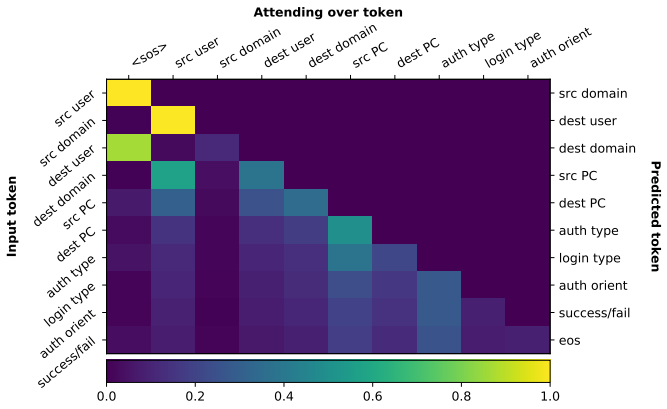

Figure 9: Average Semantic 1 attention weights.

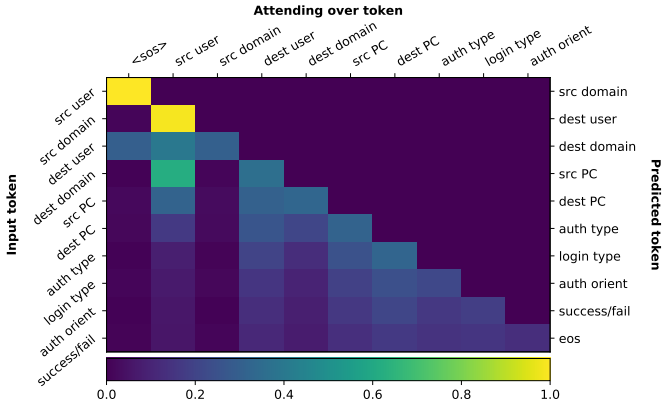

Figure 10: Average Semantic 2 attention weights.

5.1.2 Syntax. With the syntax model (Figure 8) each time step gets its own set of query weights. This makes sense for word tokenized models that have position dependent syntax. As an example of the model exhibiting intuitive behavior, when predicting the source PC, the model is attending heavily over the source user.

5.1.3 Semantic. While the semantic attention mechanisms do not assume a fixed syntactic structure, Figures 9 and 10 show that both semantic attention variants learn reasonable attention strategies on this fixed syntax data. Overall they produce similar attention maps, attending heavily to source user and source PC. Semantic 1 also attends heavily to authentication type, while Semantic 2 also deems destination user and destination PC to be important.

5.1.4 Tiered attention models. For the tiered model with a lower forward-directional LSTM, the attention weights were nearly all 1.0 for the second to last hidden state. This state is making the decision on success/fail, which conceptually makes sense with the goal of top tier LSTM to pass the most relevant information forward for the next event. Conversely, the tiered model with bidirectional LSTM cells attended fully on the very first hidden state. As Figure 5 shows, the backward LSTM ends with the first hidden state. Thus, the bidirectional tiered model is collecting both the final hidden state from the forward LSTM and the backward LSTM as its summary. This suggests that the shortcut connections attention provides are not needed for this model and task.

	1	2	3	4	5	6	7	8	9	10	
Prediction	**U22**	**DOM1**	**U66**	**DOM1**	**C1823**	**Kerberos**	**?**	**Network**	**LogOn**	**Success**	**<eos>**
True Token x(t)	**<sos>**	**U66**	**DOM1**	**U66**	**DOM1**	**C17693**	**C1966**	**NTLM**	**Network**	**LogOn**	**Success**
d(5)	0.12	0.16	0.40	0.33							
d(6)	0.04	0.04	0.27	0.17	0.49						
d(7)	0.02	0.02	0.16	0.11	0.29	0.40					
d(8)	0.03	0.03	0.13	0.10	0.24	0.34	0.13				
Prediction h(t)	**U22**	**DOM1**	**U66**	**DOM1**	**C1823**	**Kerberos**	**?**	**...**			

Figure 11: Red team word case study with semantic attention. See Figure 13 for description.

	1	2	3	4	5	6	7	8	9	10	
Prediction	**U22**	**DOM1**	**U22**	**DOM1**	**C506**	**C586**	**?**	**Network**	**LogOff**	**Success**	**<eos>**
True Token x(t)	**<sos>**	**U22**	**DOM1**	**U22**	**DOM1**	**C586**	**C586**	**?**	**Network**	**LogOff**	**Success**
d(5)	0.00	0.08	0.45	0.47							
d(6)	0.07	0.07	0.27	0.29	0.29						
d(7)	0.03	0.04	0.22	0.24	0.24	0.24					
d(8)	0.04	0.04	0.17	0.19	0.19	0.18	0.19				
Prediction h(t)	**U22**	**DOM1**	**U22**	**DOM1**	**C506**	**C586**	**?**	**...**			

Figure 12: Low anomaly word case study with semantic attention. See Figure 13 for description.

5.2 Case Studies

We consider three case studies evaluated using semantic attention models. Figures 11 and 13 depict two randomly sampled red events evaluated with word and character semantic attention models, respectively. For contrast, Figure 12 is a random non-anomalous event evaluated with the semantic word model. Tokens where the predicted and true values diverge are of significant interest as they contribute heavily to the anomaly score. We can disregard the low probabilities when predicting the source user as it is impossible to foresee what user will be associated with a random input sequence.

5.2.1 Word Tokenization. First consider the two word case studies. In both cases the source PC prediction is incorrect with low confidence. In the low-anomaly case the model is able to correctly predict the destination PC given the source PC token with very high probability. However, the red team event predicted a token associated with a different field for the destination PC. Examining the weights we see that the red team event was attending heavily over the hidden state taking the destination user domain as input and predicting the source user. We note that DOM1 is a very common domain in the LANL dataset and that the attention is likely considering the prediction that will be made from the embedding which relates to the current input token. This misclassification exposes a disadvantage in having a shared vocabulary for each field. Individual vocabularies for each field could improve performance, at the cost of minor feature engineering.

5.2.2 Character Tokenization. Finally, we examine model function when processing a character tokenized red team event. When predicting the destination PC characters the hidden state associated with the comma character right before the prediction of the source PC has the largest associated weight. The second largest weight is the comma character right before the destination PC field begins.

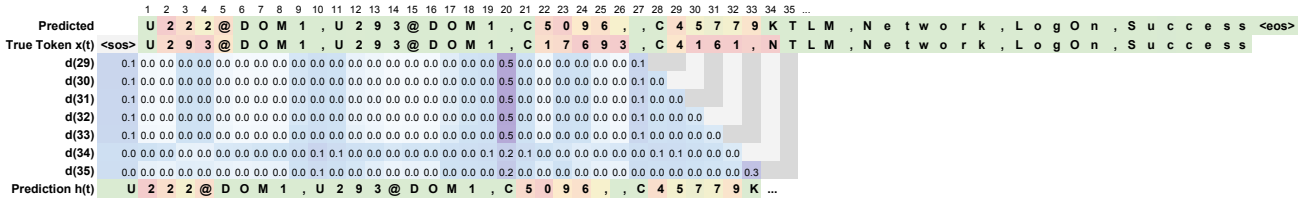

Figure 13: Red team case character study with Semantic attention. Coloring of the true token and predicted token rows is based on the probability of the given character during prediction. Green represents a near 100% probability while red is near 0%. Attention weights d(t) correspond to the top row of predictions. For example, when predicting character character 34, K, the model uses attention weights d(34). We provide a shifted copy of the predicted tokens at the bottom of the figure to align with the hidden states being attended to. Best viewed in color.

This may suggest that the model is learning positional information from the comma characters, or that it is accumulating summary vectors of the fields and "storing them" in the subsequent delimiter hidden state. Another point of interest is the attention weight vector d(34). It will substantially impact our anomaly score as our model had almost 100% confidence that the next character would be 'K', while the true token, 'N', has near 0% probability. Again we see a heavy dependence on the delimiter hidden states.

use of syntactic vs semantic attention variants

6 CONCLUSIONS

In this paper we propose five attention mechanism implementations. The fixed and syntactic attention variants can be effective for modeling sequences with a fixed structure while semantic variants are more effective for input sequences that have varying lengths and looser structures. While maintaining state-of-the-art performance, the attention mechanisms provide information on both feature importance and relational mapping between features. Additionally, architectural insights can be gleaned from the attention applied, which may in the future lead to designing more effective models. Other future work includes evaluating the system on different tasks and domains (e.g. detection of hardware failures from computer logs). One could explore additional attention variants; e.g., bidirectional models with attention may lead to further gains in performance. Finally, equipping a lower tier model with the ability to attend over upper tier hidden states, may effectively weight the relevance of previous events in a user's log sequence.

ACKNOWLEDGMENTS

The research described in this paper is part of the Analysis in Motion Initiative at Pacific Northwest National Laboratory; 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 also thank Nvidia for their donations of Titan X GPU's used in this research.

REFERENCES

[1] Guillaume Alain and Yoshua Bengio. 2016. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644* (2016).
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[3] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. 2017. Interpreting Neural Network Classifications with Variational Dropout Saliency Maps. In *Proc. NIPS.*
[4] Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. 2017. Frustratingly short attention spans in neural language modeling. *arXiv preprint arXiv:1702.04521* (2017).
[5] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proc. SIGSAC.*
[6] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Proc. NIPS.*
[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997).
[8] Alexander D Kent. 2016. Cyber security data sources for dynamic network research. *Dynamic Networks and Cyber-Security* 1 (2016), 37.
[9] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980 (2014).
[10] Zachary Chase Lipton. 2016. The Mythos of Model Interpretability. arXiv preprint arXiv:1606.03490 (2016).
[11] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).
[12] Brent Daniel Mittelstadt, Patrick Allo, Mariarosaria Taddeo, Sandra Wachter, and Luciano Floridi. 2016. The ethics of algorithms: Mapping the debate. *Big Data & Society* 3, 2 (2016).
[13] Koushik Nagasubramanian, Sarah Jones, Asheesh K Singh, Arti Singh, Baskar Ganapathysubramanian, and Soumik Sarkar. 2017. Explaining hyperspectral imaging based plant disease identification: 3D CNN and saliency maps. In *Proc. NIPS.*
[14] Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *Proc. ICASSP.*
[15] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664* (2015).
[16] Giancarlo Salton, Robert Ross, and John Kelleher. 2017. Attentive Language Models. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Vol. 1. 441–450.
[17] Aaron Tuor, Ryan Baerwolf, Nicolas Knowles, Brian Hutchinson, Nicole Nichols, and Rob Jasper. 2018. Recurrent Neural Network Language Models for Open Vocabulary Event-Level Cyber Anomaly Detection. In *Proc. AI for Cybersecurity at AAAI.*
[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. NIPS.*
[19] Alfredo Vellido, José David Martín-Guerrero, and Paulo JG Lisboa. 2012. Making machine learning models interpretable.. In *Proc. ESANN.*
[20] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.
[21] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. arXiv preprint arXiv:1502.03044 (2015).
[22] Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. 2018. Memory Architectures in Recurrent Neural Network Language Models. In *Proc. ICLR.*
[23] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *Proc. IEEE Big Data.*