2016 IEEE International Conference on Data Science and Advanced Analytics

# Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks

Adrian Taylor
*Defence R&D Canada*
*adrian.taylor@drdc-rddc.gc.ca*

Sylvain Leblanc
*Electrical and Computer Engineering Department*
*Royal Military College of Canada*
*sylvain.leblanc@rmc.ca*

Nathalie Japkowicz
*Department of Computer Science*
*American University, Washington D.C., USA*
*nathalie.japkowicz@american.edu*

***Abstract*—Modern automobiles have been proven vulnerable to hacking by security researchers. By exploiting vulnerabilities in the car's external interfaces, such as wifi, bluetooth, and physical connections, they can access a car's controller area network (CAN) bus. On the CAN bus, commands can be sent to control the car, for example cutting the brakes or stopping the engine. While securing the car's interfaces to the outside world is an important part of mitigating this threat, the last line of defence is detecting malicious behaviour on the CAN bus. We propose an anomaly detector based on a Long Short-Term Memory neural network to detect CAN bus attacks. The detector works by learning to predict the next data word originating from each sender on the bus. Highly surprising bits in the actual next word are flagged as anomalies. We evaluate the detector by synthesizing anomalies with modified CAN bus data. The synthesized anomalies are designed to mimic attacks reported in the literature. We show that the detector can detect anomalies we synthesized with low false alarm rates. Additionally, the granularity of the bit predictions can provide forensic investigators clues as to the nature of flagged anomalies.**

***Keywords*-Anomaly detection, Controller area network, Automotive, LSTM**

## I. Introduction

Automobiles have evolved from purely mechanical devices to connected computing platforms. Historically the computers in cars were isolated from the outside world, and the security of those systems was not of concern. But in recent years it has become clear that this is no longer the case; hackers have demonstrated that cars are vulnerable to cyber attacks. Such attacks work by leveraging weaknesses in the Electronic Control Units (ECUs) that control the vehicle. The ECUs communicate using the Controller Area Network (CAN) bus standard. In order to control the car, hackers typically gain access through an external interface such as cellular, bluetooth, or devices plugged into the on-board diagnostic port [1], [2]. Once they have gained a foothold, they find a way to transmit packets on the CAN bus crafted to cause specific effects. ECUs will in most cases accept properly formatted packets without authentication, making it is relatively easy to control the vehicle [3], [4]. However, any attack will modify the CAN bus traffic, and such modifications can be detected.

Attacks on the CAN bus can manifest in several ways. Diagnostic commands deployed during driving can cause malicious effects, e.g. locking the brakes to immobilize the car. However diagnostic commands should never be seen during normal driving, and so they are trivially detectable. Another tactic is to flood the bus with legitimate messages intended to cause a malicious effect. For example, some cars have a "keep lane assist" (KLA) module designed to nudge the steering wheel when the driver drifts out of his lane. The attacker can craft messages appearing to come from the KLA module that turn the wheel to drive the car off the road [4]. In order to override the legitimate sender, the attack messages must be transmitted at a high rate. Missing packets can also be evidence of an attack: if an ECU encounters enough interference when it attempts to transmit, it will eventually assume some kind of fault and go silent [5]. More recently, achieving these effects requires silencing the original ECU and sending replacement messages at the correct rate [2]. These most subtle attacks occur within an otherwise normal sequence of data words. For example, an attacker could install malware on a vehicle delivered through an mechanic's infected diagnostic tool [1]. A compromised KLA controller could be programmed to perform the steering attack described above without needing to flood the bus. The only indication of the attack on the bus would be evident in the unusual sequence of commands emanating from the KLA ECU. The commands themselves must be legitimate messages to cause an effect, so they are anomalous only in the context of recent packets from that and other devices on the bus.

We propose using a Long Short-Term Memory (LSTM) recurrent neural network (RNN) for CAN bus anomaly detection. The neural network is trained to predict the next packet data values, and its errors are used as a signal for detecting anomalies in the sequence. We chose to evaluate a LSTM predictive network on CAN bus data because it can accept raw CAN bus packet data words as an input, and we hypothesize that at a short time scale of dozens

978-1-5090-5206-6/16 $31.00 © 2016 IEEE
DOI 10.1109/DSAA.2016.20

of packets the traffic will be predictable. This method has another advantage, in that it requires no domain knowledge of the system it is modelling. On the CAN bus this means the data messages need not be decoded. Such an advantage is doubly important in the automotive domain, where the protocols are different for each manufacturer, model, and even from year to year on the same model. The LSTM-based anomaly detector can be applied to many different vehicles without substantial modification.

The remainder of this paper is organized as follows. We review related work in anomaly detection and CAN bus attack detection in Section II. In Section III we describe the automotive CAN data used in this research and how we produced anomalous data mimicking those attacks. The LSTM-based detector is described in Section IV. In Section V we present our experiments and results, with our conclusions in Section VI.

## II. Related Work

### A. Anomaly Detection in Sequences

A variety of approaches have been applied to anomaly detection in sequences [6]. A common approach is to identify and compare patterns in $n$-grams, where an $n$-gram is a length-$n$ subsequence. Anomalies can take the form of foreign symbols, rare $n$-grams, or foreign $n$-grams [7]. Many algorithms measure the distance between candidate $n$-grams and historical $n$-grams or other parts of the same sequence, e.g. Keogh's discord-finding methods [8]. These methods are difficult to apply to CAN traffic because they assume a finite symbol dictionary, or use quantization methods to convert continuous values to a finite set of symbols. CAN symbols are 64-bit binary words. The number of possible words for a particular ID in a 64-bit address space is potentially infinite over the lifetime of the car, so we cannot rely on foreign symbols or $n$-grams to detect anomalies. Converting CAN data into a format suitable for distance-based anomaly detection methods would require the development of compression algorithms analogous to SAX [8]. Their performance will then be dependent on the quality of those algorithms. Similarly Hidden Markov Models could be used for CAN bus packets, but they would need to be adapted for 64-bit data or operate on a compressed representation of the data. Instead, RNNs are a more natural fit for CAN bus data sequences.

LSTM-based RNNs have recently been shown to be very effective for anomaly detection in standard time series test sets [9], electrocardiography [10], and aircraft telemetry [11]. Each of these systems trains a RNN to predict the next symbol in the sequence. Anomalies are flagged when the error between real and predicted values is high. When the series is inherently unpredictable, the LSTM encoder-decoder can produce superior results to the LSTM predictor [12]. Variations using energy-based models [13] and variational inference [14] has also been presented, although without direct comparisons on the same data it is not clear under what circumstances one might be superior to another. We based our detector on the original LSTM structure because it has been shown to be robust in a variety of domains.

### B. CAN Bus Anomaly Detection

Previous approaches to detecting attacks on the CAN bus have mainly been based on timing information. CAN packets are normally transmitted at a regular frequency, motivating frequency detection as a defence against most attacks [4], [15]; such methods have been shown to be very effective at detecting inserted or missing packets [16], [17]. A different time-based method uses clock skew to authenticate ECUs [18]. Many attacks originate from an unusual source: either a foreign ECU, or a compromised one that normally sends different messages. An ECU sending malicious packets can be identified because its clock skew is different from the legitimate ECU. However this method could be defeated by an adversary able to match clock skews, and it cannot detect attacks when the compromised ECU is the normal emitter of the IDs of the malicious packets. Fewer methods have investigated packet data contents directly. Packet message entropy was used to detect insertion attacks [19], but the method was not evaluated against attacks that affect only the message contents of a packet stream. Another approach automatically classifies fields in CAN messages and measures valid ranges based on previous data [20]. However it was not evaluated with any attack scenarios. Our LSTM detector is the only approach that can detect packet data anomalies that are unusual only in the context of the rest of the sequence.

## III. Data

### A. CAN Bus Data

Our work is based on data obtained from a 2012 Subaru Impreza high speed bus. We captured approximately 19 hours of driving data, obtained with a USB-CAN bus interface connected to the in-car on-board diagnostics (OBD-II) port. The bus transmits at 500 kbps. Twenty distinct IDs were observed (for convenience we identify the IDs by the numerals 0 through 19 and not by their actual values). About 800 packets per second are transmitted while the car is running, with each ID transmitting at a consistent period varying from 10 ms to 1s; this amounted to approximately 45 million packets in our data set.

Each ID uses only a subset of its 64 bits. Many bits were observed to always be zero, and some always one. The average values of each bit for each message is illustrated in Fig. 1. We make use of this information in generating anomalies to test our anomaly detector (see Section III-B below). However we made no attempt to decipher the traffic and assume no knowledge of its meaning.

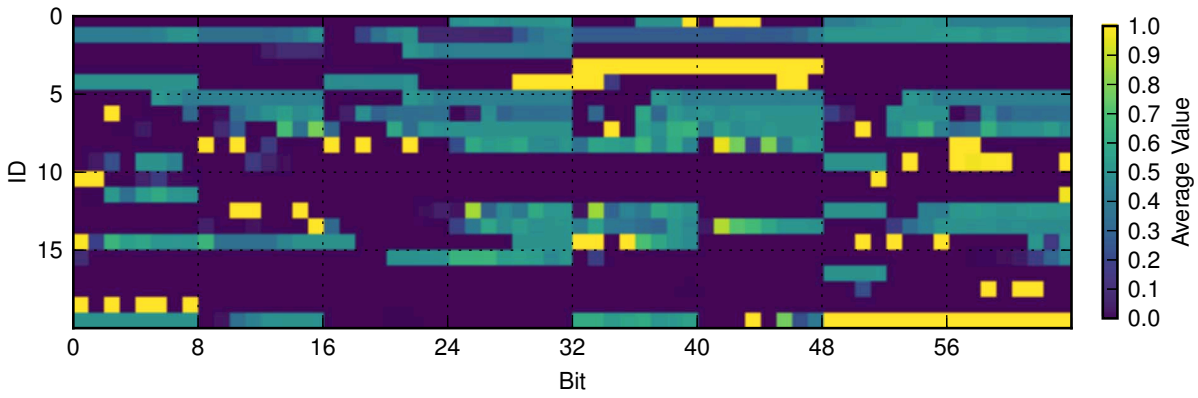



Figure 1. Average value of each bit for every ID.

We detect anomalies in each ID's individual packet stream independently. However not al IDs produce an abundance or variety of packet data. Of the 20 IDs, one (18) contained no data, and two more (3, 17) had very little data variation. All three were excluded from our experiments. Other IDs produced a much larger proportion of unique data words, making up 10 to 100% of the observed packets. Almost all the IDs produced new words at a steady rate over the full length of our data capture. This fact motivated the requirement for a sequence anomaly detector that is not restricted to operating on a fixed dictionary of symbols; on the CAN bus, if such a dictionary exists, it is not apparent within the data available to us. The rate of novel words could be caused by the inherent variation in physical processes in the vehicle, or by the combination of those factors with artificial variation such as counters embedded in the messages.

### B. Synthesizing CAN Bus Anomalies

Unfortunately there is no publicly available repository of attack traffic, so we resort to synthesizing anomalies by modifying packet capture data. Creating actual attack effects on a car is time consuming and requires specialized knowledge, and the willingness to risk damage to the test vehicle and its surroundings [4]. Consequently most researchers resort to emulating or simulating effects characteristic of the attacks they are attempting to detect, e.g. switching message IDs to simulate a counterfeit transmitter [18], or inserting innocuous random packets into the live bus [17].

We take a similar approach. High-speed CAN traffic is highly regular, so attacks manifest in only three basic ways: packets are added, expected packets do not appear, or the data contents of the packets are unusual. Because we are not using any timing information in our attack scenarios, we can create attack signatures just by altering the sequence of packets from our data captures. By altering only the sequences, we guarantee that each packet is valid; attack traffic must contain legitimate messages, otherwise it would not achieve any effect. Thus unusual data is anomalous only in the context of neighbouring packets. The only exception is a case where unused bits are flipped to simulate a hidden command and control channel.

We produced five kinds of anomalies:

- *Interleave*. Subsequences are constructed by interleaving two normal subsequences from different points in time. This case simulates a situation where two senders are sending competing commands on the bus at the same rate. Most published attacks are of this nature.
- *Drop*. A normal subsequence has three words removed in the middle. This simulates an attack where an ECU is suppressed, and quickly replaced by an impersonator sending (for the time being) normal packets.
- *Discontinuity*: Subsequences are constructed by drawing the first half from one one period of time, and the second half from another point in time much later. This jump simulates an attack where a compromised ECU suddenly switches from sending normal traffic to some out of context, but otherwise legitimate traffic. The discontinuity case is the most important one in that it can only be detected by analyzing packet data contents, and would be invisible to an anomaly detector looking solely at packet timing information.
- *Unusual*. As shown in Section III, most IDs do not use all 64 of their bits. In the *unusual* case, two of these normally unused bits are flipped from zero to one. This case simulates the use of a hidden command and control channel embedded in normal traffic.
- *Reverse*. Subsequences are modified to run in reverse order. While we cannot guarantee that the resulting subsequence is anomalous, but in most cases it is unlikely to be similar to normal sequence data. This case does not correspond to any attacks; it is used as a baseline for comparison with normal data and the other cases.

Each anomaly case serves a separate purpose. The *drop* and *interleave* cases mimic packet insertions and erasures that are the basis of many reported attacks. They are identical to some of the simulated attacks presented in previous work [16], [17]. These cases permit comparison of our detector with other methods. The *discontinuity* case simulates a more sophisticated attack similar to the one presented by Valasek and Miller [2]. A version of this attack was simulated by Cho and Shin [18], but in that work only the sender's identity was determined. Here the *discontinuity* case creates an abrupt shift from one sequence of valid messages to another. The *unusual* case evaluates our detectors ability to learn basic patterns of bit use in the individual messages. This kind of attack would not be detectable without monitoring the bits within each message in a sequence. We have not seen evidence of this case evaluated by previous detectors. Finally, the *reverse* case is intended to evaluate how our detector deals with 'obvious' anomalous traffic.

## IV. LSTM Neural Network for CAN packet anomaly detection

### A. LSTM Recurrent Neural Networks

Our anomaly detector is a RNN with LSTM units [21]. The basic RNN is a neural network with feedback (Fig. 2). The output $h_t$ is a function of both the input $x_t$ and the previous output $h_{t-1}$:

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

where $W, U$ are weight matrices, $b$ is a bias term, and $f$ is a nonlinear transformation (e.g. the sigmoid or tanh function). In this way RNNs make use of information from both the past and the present. However they are difficult to train. Neural networks are trained with backpropagation, a process of adjusting the weights according to the derivative of the output errors. The feedback loop in RNNs cause errors to shrink or grow exponentially, destroying the information in the backpropagation signal. Hochreiter solved this gradient problem by introducing additional structure to the network in the form of explicit input and forget gates [21]. These gates depend on both the input $x_t$ and previous step's output $h_t$: (Fig. 3), and control how much of the internal state depends on the new input and the previous state.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

Here $W_i, U_i, W_f, U_f$ are learned weights, and $b_i$ and $b_f$ are learned bias terms. The sigmoid function $\sigma$ is typically used for the gate terms to scale their output between 0 and 1. The input and forget signals ($i_t$ and $f_t$ determine the new cell state $c_t$ as a linear combination of the previous internal state

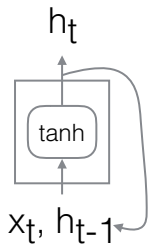



Figure 2. Basic RNN cell. The weight multiplications are omitted.

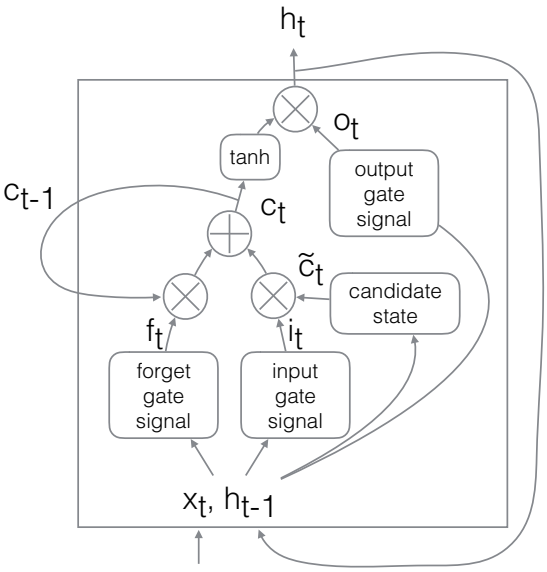



Figure 3. LSTM basic unit. The LSTM adds additional connections designed to facilitate forgetting and remembering the past. Note the weight multiplications have been omitted for clarity in emphasizing the information flows.

$c_{t-1}$ and new candidate internal state $\tilde{c}_t$:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t c_{t-1} + i\tilde{c}_t$$

where again $W_c$ and $U_c$ are weight matrices and $b_c$ is the bias. The output is also controlled by a gate function with its own weights $W_o, U_o$ and bias $b_o$:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$h_t = o_t \tanh(c_t)$$

The design and evaluation of LSTM variants is an active area of research; we chose the standard structure because it has been proven to work well over a range of problem domains [22].

### B. Network Architecture

The basic structure of our network is similar to previous LSTM anomaly detectors, e.g. [9], but the input in the CAN domain is a high dimensional binary vector. Inspired by

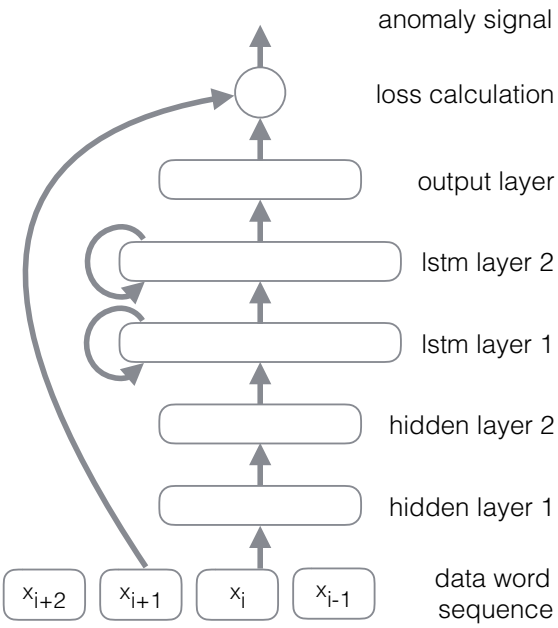



Figure 4. LSTM-based anomaly detector structure.

an approach used in natural language processing [23], we experimented with adding linear embedding layers to project the binary inputs into a real-valued state space space. Our final network is composed of a linear segment, a recurrent segment, and a linear output segment (Fig. 4). Given an input sequence $\{x_1, x_2, x_3, \ldots, x_N\}$, where $x_i$ is a 64-bit vector, the network is trained for each $i$ with a target of the subsequent word $y_i = x_{i+1}$. At each time step, the input $x_i$ is the 64-bit data word from the packet. These bits are transformed by two non-recurrent hidden layers, each with 128 units and tanh activation functions. The output of the linear layers is fed into two recurrent LSTM layers, each with 512 units and tanh activation functions. The final layer is again linear, with 64 units using sigmoid activation functions to scale the final values to between 0 and 1. A separate network was trained for each CAN bus ID.

This network structure was selected by experimenting with different numbers of layers, hidden units, activation functions, and other parameters and evaluating with validation data. The final network was trained with a dropout probability of 0.2 at each layer, including the input, as a regularization method to prevent overfitting. The training data was composed of subsequences 20 words long, presented in batches of size 128. Training was stopped when the validation performance failed to improve after five consecutive runs through the training data.

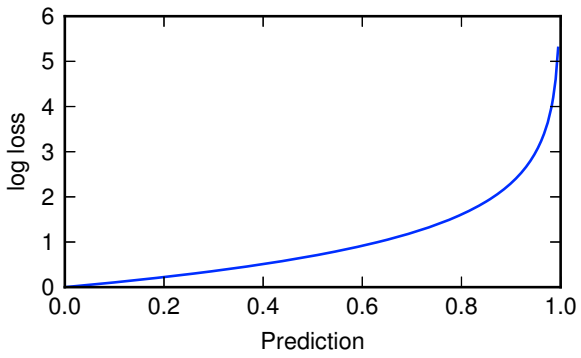



Figure 5. Log loss for target of 0.

### C. Log Loss Error

The error signal for the network is the binary log loss between the output and the next data word in the sequence. The binary log loss is defined as:

$$L(\hat{b}_k, b_k) = -\Big(b_k \log(\hat{b}_k + \epsilon) + (1 - b_k) \log(1 - \hat{b}_k + \epsilon)\Big)$$

where $b_k$ is the $k^{\text{th}}$ bit in the target $y_i$, $\hat{b}_k$ is the $k^{\text{th}}$ bit's predicted value by the network at step $i$, and $\epsilon$ is a fixed value that caps the maximum loss. Fig. 5 plots the log loss for a target of 0.0 (the plot is symmetrical for a target of 1.0). The log loss function is low for incorrect but middling predictions, but very high for confident but incorrect predictions. It approaches infinity for a perfectly incorrect prediction; in practice, it is capped at a maximum value of $-\log(\epsilon)$. The experiments reported here use $\epsilon = 10^{-15}$, which translate to a maximum bit loss of approximately 35. The log loss error encourages the network to quantify its uncertainty by penalizing unwarranted confidence.

### D. Anomaly signal

The log loss on each bit is also the basis of the anomaly signal. Its property of highly penalizing confidently incorrect decisions of the network is a way to measure surprising outcomes. We combine the bit losses over an entire test sequence to produce a scalar anomaly score for the entire sequence. The method of combining values must balance the noisiness of individual bit losses against the importance of a single bit's score as an indicator of an anomaly. We compared several ways of combining the bit losses over the entire sequence:

- *Maximum bit loss*: the maximum of all bit losses over the entire test sequence.
- *Maximum word loss*: Define the word loss as the average bit loss for a word. Take the maximum of all word losses over the entire test sequence.
- *Window max*: Calculate a moving average of the five most recent word losses over the sequence. Take the maximum of the window averages.

- *Log window max*: Calculate a rolling window of the logarithmic mean of the five most recent word losses. Take the maximum of the window logarithmic averages.
- *Sequence mean*: Calculate the mean bit loss over the entire test sequence.

Different methods of combining bit losses yield different tradeoffs in performance; details are presented in the results below.

## V. Experiment and Results

### A. Training and Testing Data

We captured approximately 19 hours of data, which contained on the order of 50 million packets. Approximately 13 hours of this data was used for training. Two of the hours were reserved for model selection and early stopping (validation). The final four hours of data was divided into two test sets.

Each test set was divided into six equal parts. One was left unmodified as normal test data, and the other five were modified to simulate different kinds of anomalies as detailed in Section III-B, yielding six test cases. For each test case, the data was divided into subsequences with 100 steps. Each ID's data packets was treated as an independent sequence, generating a complete experiment with its own training, validation, and test data. A different LSTM network was trained for each ID's data, and all testing and evaluation was also repeated for each one.

### B. Performance Measure

Performance is evaluated using the Receiver Operating Characteristic (ROC) curve. A ROC is a plot of the false positive rate (FPR) against the true positive rate (TPR) as the detection threshold is varied. ROC curves provide a way to compare different methods that is independent of the decision threshold. The ideal discriminator is at the top left of the graph, with 100% detection and zero false-positives. The diagonal of the graph represents random guesses and is the worst possible case. To summarize ROC performance, we used the area under curve (AUC), which is simply the area under a ROC curve. The AUC condenses the curve to a single number, which makes it easier to compare a large number of results. However the AUC can also obscure information about performance tradeoffs; details for individual experiments are discussed below. In each test case, the same normal data for a given ID was used to calculate the FPR to ensure a fair comparison of performance between anomaly types.

### C. Anomaly signal comparison

The method of combining bit losses to generate an anomaly signal has a strong effect on the resulting TPR and FPR. Fig. 6 shows the ROC curve for the *discontinuity* test case using each method of combining bit losses as described in Section IV-D. Only the bottom 1% of the graph is shown.

Table I
LSTM Predictor AUC using *Maximum Bit Loss*

| ID | discontinuity | drop | interleave | reverse | unusual |
|---|---|---|---|---|---|
| 0 | 0.999651 | 0.999184 | 0.999044 | 0.999046 | 1.0 |
| 1 | 0.943856 | 0.493971 | 0.967228 | 0.620695 | 1.0 |
| 2 | 0.998940 | 0.520689 | 0.997237 | 0.635042 | 1.0 |
| 4 | 0.999808 | 0.998883 | 0.999734 | 0.997898 | 1.0 |
| 5 | 0.999642 | 0.525107 | 0.992955 | 0.850504 | 1.0 |
| 6 | 0.999981 | 0.998928 | 1.000000 | 0.999516 | 1.0 |
| 7 | 0.999794 | 0.638822 | 0.999062 | 0.911690 | 1.0 |
| 8 | 0.924524 | 0.187951 | 0.660048 | 0.846570 | 1.0 |
| 9 | 0.997855 | 0.996505 | 0.999988 | 0.990238 | 1.0 |
| 11 | 0.994629 | 0.547290 | 0.995878 | 0.350318 | 1.0 |
| 12 | 0.992308 | 0.992140 | 0.999626 | 0.993701 | 1.0 |
| 13 | 0.997034 | 0.455347 | 0.995219 | 0.583802 | 1.0 |
| 14 | 0.994547 | 0.989593 | 1.000000 | 0.987090 | 1.0 |
| 15 | 0.997801 | 0.997077 | 0.999631 | 0.998810 | 1.0 |
| 16 | 0.995885 | 0.995827 | 0.995824 | 0.994475 | 1.0 |
| 19 | 0.324561 | 0.472222 | 0.176471 | 0.674383 | 1.0 |

There is no one best method; the choice will depend on whether we wish to minimize the FPR or maximize the TPR. If we wish to maximize the TPR, the *maximum bit loss* method achieves 100% detections at the lowest FPR. But if we minimize the FPR by choosing a threshold that yields no false positives, the *window max* method is significantly better. This pattern holds across nearly all the IDs.

### D. Overall results

AUC results for all test cases are shown in Table I. We report AUC using the *max bit loss* anomaly signal because it gives the best overall performance in each case. The most important case is *discontinuity*, because it is the one that can only be detected by looking at the bits in each data message. For the majority of IDs, the AUC is very high, but there are exceptions. For IDs 8, 11, and 13, there are cases with an AUC below 0.5. Such AUC scores indicate that anomaly traffic is generating lower loss in the detector than normal traffic. If the AUC is around 0.5, the detector has failed to internalize the message structure, or more likely that it is inherently unpredictable. If it is well below 0.5, as for ID 19, then the detector is actually reliable at making errors – its decisions could be reversed to obtain better performance. This curious behaviour merits further investigation; however it is difficult to do so without an understanding of the meaning of the bits.

Additionally, we verified that our problem is not trivially simple by evaluating a predictor that guesses the current word at each time step. If bit values tended to change infrequently, this approach could do reasonably well, and the LSTM detector might be using the same strategy. Table II shows the trivial detector AUC using the *maximum word loss* measure for all test cases, which dominated other method AUCs overall. In almost all cases performance is substantially inferior to the LSTM detector; in many cases it is no better than chance. There are some exception, e.g. ID 11 which has similar AUC values to the LSTM method

Table II
TRIVIAL PREDICTOR AUC USING *Maximum Word Loss*.

| | word_max discontinuity | drop | interleave | reverse | unusual |
|---|---|---|---|---|---|
| 0 | 0.704800 | 0.462238 | 0.836322 | 0.512256 | 0.597934 |
| 1 | 0.775631 | 0.462487 | 0.947713 | 0.492929 | 0.555081 |
| 2 | 0.700612 | 0.440628 | 0.823718 | 0.549786 | 0.680094 |
| 4 | 0.522282 | 0.466723 | 0.751193 | 0.512343 | 0.664052 |
| 5 | 0.784174 | 0.442769 | 0.924237 | 0.539426 | 0.546287 |
| 6 | 0.900386 | 0.445727 | 0.978563 | 0.571334 | 0.571154 |
| 7 | 0.957976 | 0.449022 | 0.994164 | 0.535728 | 0.568417 |
| 8 | 0.816481 | 0.520340 | 0.752540 | 0.519495 | 0.761675 |
| 9 | 0.566447 | 0.497651 | 0.603905 | 0.495580 | 0.997470 |
| 11 | 0.995287 | 0.495626 | 0.994689 | 0.500953 | 0.997294 |
| 12 | 0.363417 | 0.135182 | 0.621624 | 0.159113 | 0.076946 |
| 13 | 0.948003 | 0.452410 | 0.999976 | 0.598184 | 0.659048 |
| 14 | 0.710125 | 0.470280 | 0.912983 | 0.477558 | 0.734806 |
| 15 | 0.486674 | 0.470139 | 0.775697 | 0.530541 | 0.683187 |
| 16 | 0.500000 | 0.500000 | 0.500000 | 0.500000 | 1.000000 |
| 19 | 0.513158 | 0.508041 | 0.654412 | 0.492284 | 0.628268 |

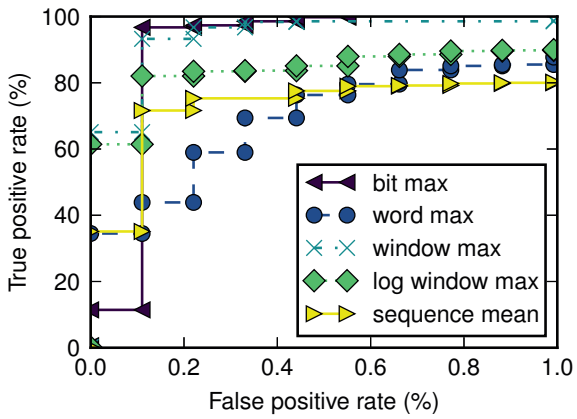



Figure 6. *Discontinuity* ROC curves for ID 2 using different anomaly signals.

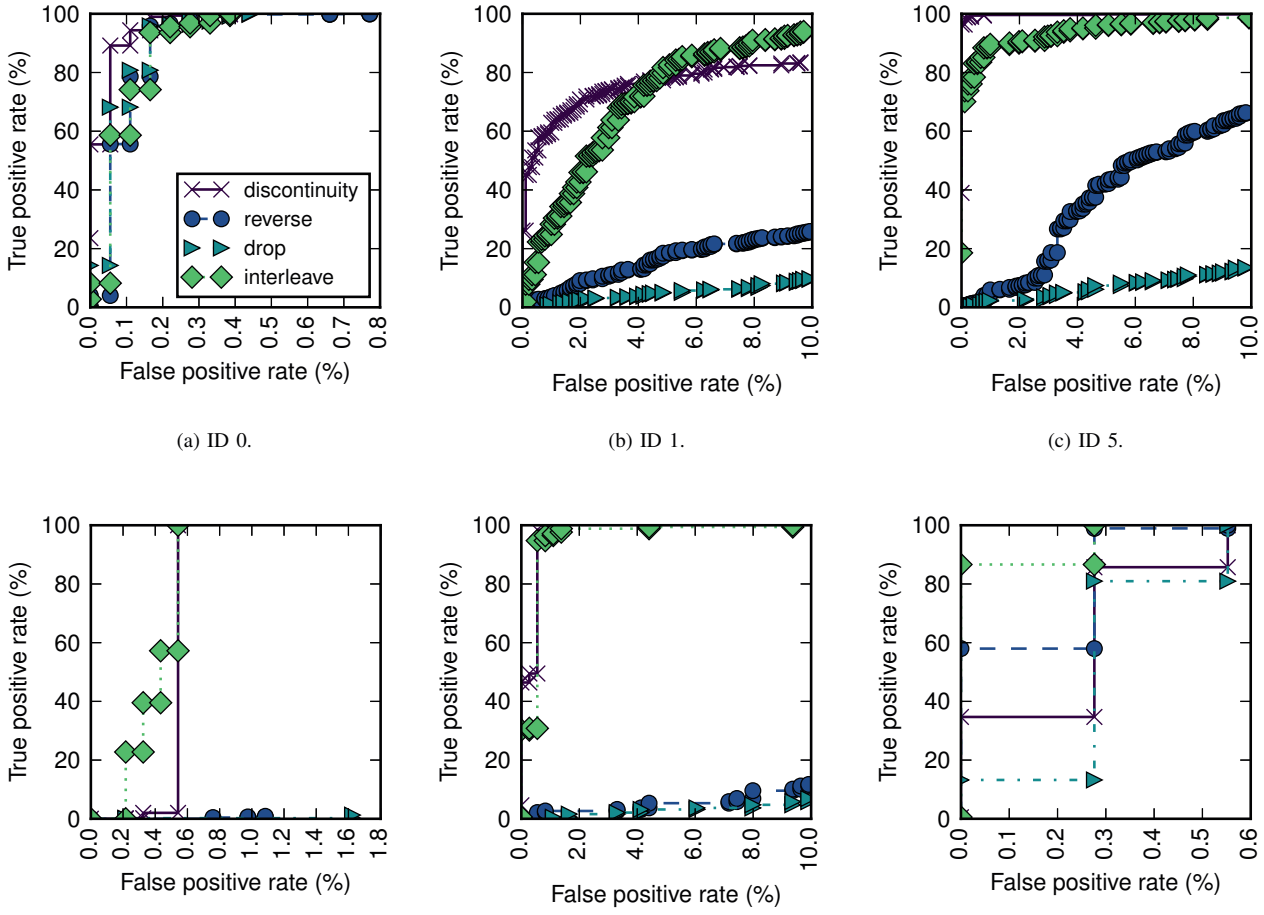
*maximum bit loss*. Upon further investigation we determined ID 11's data sequence is mainly composed of the same value repeated for long periods of time. Word changes are rare, so the anomaly cases that cause changes to occur are very different from the normal case. Other IDs that have high AUC for the trivial detector likely have similar explanations. Since one of our goals is to avoid having to learn the details of data sequences, we did not investigate every case and are content that the LSTM detector performance is significantly better than that of the trivial one. This performance verifies that for most IDs, the patterns in the data sequences have non-trivial complexity.

The complete ROC curve for each case allows more granular performance analysis. Fig. 7 shows the ROC curves for a selection of IDs (for IDs 0, 1, 5, 11, and 13) and test cases (the *unusual* case is omitted because its performance is perfect). Again we focus on the *max bit loss* signal. Each graph shows only the portion of the ROC curve where the FPR is below 10% (for IDs 0 and 11 the curve reaches 100% TPR more quickly; only the relevant segment is shown). This cross-section of results illustrates how the shape of the ROC curve influences performance. Each of the IDs shown has a high AUC, but for practical purposes performance varies dramatically. ID 1, for example, has the lowest AUC (94.4), but at FPR 0% yields a TPR of about 22% for the *discontinuity* case. By contrast, the ID 11 detector has an AUC of .998, but cannot detect any discontinuities at a 0% FPR. The other cases are examples of good but not perfect performance. For most IDs, performance is in fact quite good, except at the lowest FPRs. The ID 0 ROC illustrates the difficulty of the problem. Each step in the graph's shape corresponds to an outlier high-loss value in the normal data. Thus the difference between 55% and 90% *discontinuity* detections is a single bit's unusually high loss in the normal data. Measuring performance at this level could require substantially more data. Improving performance may require better understanding of the nature of the outliers in the normal data.

### *E. Practical thresholds and performance*

For evaluating the system in terms of practical use, we must choose a threshold for the detector and determine the resulting FPRs and TPRs. An unsupervised anomaly detector in a car must have a false alarm rate of effectively zero. For example, consider a system that evaluates traffic every half second. A false alarm rate of $10^{-4}$ will produce a false alert every hour. A rational user will quickly learn to ignore the detector, rendering it useless. Consequently we first focus on detection performance with a zero false alarm rate for each ID.

If we select a threshold that yields a false alarm rate of zero in the test data, what is the resulting TPR for each test case? We cannot guarantee that this threshold choice will prevent all false alarms, but it provides a realistic estimate of detection performance at the limit of low false alarm rates. For each ID, we chose a threshold that gave the highest TPR for a zero FPR. We generated a second set of test cases using the same method as for the first, but with new CAN bus captures from the same vehicle. We captured an additional two hours of driving data under similar conditions as the first 17 hours, and generated test cases from the new data using the same procedure as the initial test cases. Table III shows the TPR for each new test case using the threshold selected from the original test cases for no false positives in the *discontinuity* test case. As explained above, the *max window* measure gives the best performance at low FPRs so the table shows results using this method. Again, we focus on the discontinuity case because other kinds of anomalies can all be detected with other methods. Performance ranges from zero to 100% detections. For three IDs, the system achieves nearly perfect detection; in five more the detection rate is well over 90%.




(a) ID 0. (b) ID 1. (c) ID 5.

(d) ID 11. (e) ID 13. (f) ID 15.

Figure 7. ROC curves for a selection of IDs, illustrating the trade-offs in true and false positive rates for each test case.

Whether this performance is sufficient for practical use depends on external factors. For example, the frequency of an event for a successful attack is a factor; a less than perfect detection rate may be acceptable of the attack is expected to occur multiple times, increasing the overall likelihood that it is detected (assuming the probabilities of detecting each event are independent). Furthermore, if the requirement that a FPR of 0% is relaxed, we can reach much better detection rates.

To get a sense of the upper limit of anomaly detection performance, we reverse our requirement for zero false alarm rate and instead find a threshold that yields 100% TPR. Again we selected a threshold for 100% true positive and evaluated the FPR using a second test set. The resulting FPR is shown in Table IV. The table has only one column because the same normal data is used for each anomaly case. Again the threshold is based on the *discontinuity* case only. But here the *max bit* score is used, because for most IDs

Table III
TPR WITH NO FALSE POSITIVES.

| ID | discontinuity | drop | interleave | reverse | unusual |
|---|---|---|---|---|---|
| 0 | 0.485390 | 0.003089 | 0.983094 | 0.314114 | 0.004154 |
| 1 | 0.545455 | 0.001546 | 0.971235 | 0.037931 | 0.020790 |
| 2 | 0.409091 | 0.000000 | 0.411168 | 0.000000 | 0.000000 |
| 4 | 0.944805 | 0.044822 | 1.000000 | 0.674138 | 0.002079 |
| 5 | 0.922078 | 0.000000 | 0.983080 | 0.127586 | 0.006237 |
| 6 | 0.907468 | 0.004633 | 1.000000 | 0.061102 | 0.000000 |
| 7 | 0.844156 | 0.000000 | 0.947591 | 0.001721 | 0.000000 |
| 8 | 0.444805 | 0.000000 | 0.297800 | 0.000000 | 0.002079 |
| 9 | 0.852564 | 0.484067 | 1.000000 | 1.000000 | 0.069815 |
| 11 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 12 | 0.669355 | 0.646388 | 1.000000 | 0.991489 | 0.072165 |
| 13 | 0.991870 | 0.000000 | 1.000000 | 0.004310 | 0.000000 |
| 14 | 0.991870 | 0.000000 | 1.000000 | 0.142241 | 0.104167 |
| 15 | 0.495935 | 0.154440 | 1.000000 | 0.831897 | 0.072917 |
| 16 | 0.508197 | 0.937984 | 1.000000 | 1.000000 | 1.000000 |
| 19 | 1.000000 | 0.192308 | 1.000000 | 1.000000 | 1.000000 |

Table IV
FPR REQUIRED FOR 100% TPR

| ID | FPR |
|---|---|
| 0 | 0.0048 |
| 1 | 0.6308 |
| 2 | 0.0048 |
| 4 | 0.0039 |
| 5 | 0.0998 |
| 6 | 0.0010 |
| 7 | 0.0044 |
| 8 | 0.5698 |
| 9 | 0.0096 |
| 11 | 0.0010 |
| 12 | 0.0287 |
| 13 | 0.0413 |
| 14 | 0.0049 |
| 15 | 0.0024 |
| 16 | 0.0098 |
| 19 | 0.6341 |

the *max bit* ROC curves have a 100% TPR at the lowest FPR of all the scoring methods. In most cases, the FPR on the *discontinuity* case is relatively low: in seven cases it is less than 0.01, in five above that but below 0.1, and in only five cases is it above 0.1. These results are very close to the required performance level. If a non-zero false alarm rate could be mitigated in some way, this method could be effective and reliable for detecting bit field anomalies in the CAN data stream.

### F. Comparison with Previous Work

The frequency methods discussed in Section II-B have been shown to detect attacks corresponding with our *drop* and *interleave* cases with perfect accuracy, and so are superior to our method for these scenarios. The *unusual* bit use case could be detected by monitoring the normal values of bits and flagging previously unseen values; in theory the method presented in [20] could perform as well as our method, although it was not evaluated with actual anomalies. Our method found these with perfect accuracy. The *discontinuity* test case would be invisible to those methods, but could be detected with clock skew fingerprinting if the new packet stream originates from a new ECU. The clock skew authentication method can detect any anomalies that arise from an unexpected sender of packets with a reported perfect accuracy. However if the rogue packets originate from the same ECU, or the new ECU is able to mimic the clock drift of the original sender, only our method will detect this case.

The best solution may be a combination of methods. Frequency-based sensors would identify packet insertions, fingerprinting or other methods would identify foreign transmitters, packet inspectors could identify malformed packets, and the LSTM method described here would identify sequence context anomalies.

### G. Bit Loss for Attack Forensics

We observe that the granular nature of the LSTM predictor's loss output is of additional use in anomaly forensics. The LSTM predictor produces a loss for each individual bit; thus, given an anomalous packet trace, the suspect bits of each data message are specifically identified. An engineer familiar with the protocol for the car could use such information to help diagnose the nature of each anomaly. Additionally, if anomaly examples become available, the detector could use this meta-pattern of loss to classify them according to their loss signatures.

## VI. CONCLUSIONS

We have demonstrated the detection of sequence data anomalies on the automotive CAN bus using LSTM neural networks. The LSTM approach has the advantages of not requiring knowledge of the specific protocol, and has shown promising performance in detecting a range of anomalies corresponding to known attacks on the CAN bus, and can detect a kind of attack invisible to previous detection methods. However more work needs to be done to achieve a practical level of false alarm rate while reliably detecting anomalies. One shortcoming of the current approach is that it treats each ID's data sequence as independent. It is more likely that there are data interdependencies between IDs, so that a detecter that monitors all IDs simultaneously may achieve better overall performance. We must also compare our approach with other sequence anomaly detection methods, e.g. Hidden Markov Models. Additionally we hope to create opportunities to test our methods on real attack data.

An anomaly detector installed in a vehicle must be able to adapt to changes in normal behaviour over the lifetime of the car. Changes in the bit patterns may result from firmware upgrades, replaced components, and the wear and tear of physical components resulting from normal driving. The anomaly detector must be able to learn from online data while still detecting anomalies. Therefore online learning will also be a necessary part of future work.

Finally, a practical anomaly detector does not live alone but is part of a security ecosystem. The LSTM-based detector could be one kind of sensor, working together with frequency-based detectors, packet authentication, and other security devices. In the age of connected vehicles, there could be a monitoring component for a fleet of cars. Such an approach could contribute to managing the false alarm rate by building knowledge of normal outliers for each make and model of car. The bigger picture also addresses an issue we avoided, the question of how to respond to an anomaly detection. For example, a high-confidence alert could automatically put the vehicle into a "safe mode" designed to allow the driver to safely stop the car while preventing further damage from being done. Alerts could also be transmitted to the manufacturer for further analysis. Regardless of how system designers incorporate an anomaly detector, if it can be made reliable it will become an important part of automotive cyber security.

## REFERENCES


[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Security Symposium 2011*. Berkeley, CA, USA: USENIX Association, 2011.

[2] C. Valasek and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," IOActive Labs Research, Tech. Rep., 2015. [Online]. Available: http://www.ioactive.com/labs/resources-white-papers.html

[3] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *Proc. IEEE Symposium on Security and Privacy 2010*, 2010, pp. 447–462.

[4] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," IOActive Labs Research, Tech. Rep., Aug. 2013. [Online]. Available: http://blog.ioactive.com/2013/08/car-hacking-content.html

[5] A. G. Illera and J. V. Vidal, "Dude, WTF in my CAN!" presented at Black Hat Asia 2014. [Online]. Available: https://www.blackhat.com/asia-14/briefings.html

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection for Discrete Sequences: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 5, pp. 823–839, May 2012.

[7] R. Maxion and K. Tan, "Anomaly detection in embedded systems," *IEEE Trans. Comput.*, vol. 51, no. 2, pp. 108–120, Feb. 2002.

[8] E. Keogh, J. Lin, and A. Fu, "HOT SAX: efficiently finding the most unusual time series subsequence," in *Proc. Fifth IEEE International Conference on Data Mining*, Nov. 2005, pp. 226–233.

[9] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *Proc. 23rd European Symposium On Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, 2015, pp. 89–94.

[10] S. Chauhan and L. Vig, "Anomaly detection in ECG time signals via deep long short-term memory networks," in *Proc. IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015. 36678 2015*, Oct. 2015, pp. 1–7.

[11] A. Nanduri and L. Sherry, "Anomaly detection in aircraft data using Recurrent Neural Networks (RNN)," in *Proc. 2016 Integrated Communications Navigation and Surveillance*, Apr. 2016, pp. 5C2–1–5C2–8.

[12] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection," in *Presented at ICML 2016 Anomaly Detection Workshop*, New York, NY, Jul. 2016. [Online]. Available: https://arxiv.org/abs/1607.00148

[13] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep Structured Energy Based Models for Anomaly Detection," in *Proc. 33rd International Conference on Machine Learning*, 2016, pp. 1100–1109.

[14] M. Soelch, J. Bayer, M. Ludersdorfer, and P. van der Smagt, "Variational Inference for On-line Anomaly Detection in High-Dimensional Time Series," *arXiv:1602.07109 [cs, stat]*, Feb. 2016, arXiv: 1602.07109. [Online]. Available: http://arxiv.org/abs/1602.07109

[15] C. Miller and C. Valasek, "A Survey of Remote Automotive Attack Surfaces," IOActive Labs Research, Tech. Rep., 2014. [Online]. Available: http://www.ioactive.com/labs/resources-white-papers.html

[16] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *Proc. 2015 World Congress on Industrial Control Systems Security (WCICSS)*, Dec. 2015, pp. 45–49.

[17] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *Proc. 2016 International Conference on Information Networking (ICOIN)*, Jan. 2016, pp. 63–68.

[18] K.-T. Cho and K. G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *Proc. 25th USENIX Security Symposium*, Austin, TX, USA, Aug. 2016.

[19] M. Muter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proc. 2011 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2011, pp. 1110–1115.

[20] M. Markovitz and A. Wool, "Field Classification, Modeling and Anomaly Detection in Unknown CAN Bus Networks," in *escar Europe 2015*, 2015. [Online]. Available: https://www.escar.info/history/escar-europe/escar-europe-2015-lectures-and-program-committee.html

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[22] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2016.

[23] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM Neural Networks for Language Modeling." in *INTERSPEECH-2012*, 2012, pp. 194–197.