Smart Anomaly Detection: Deep Learning modeling Approach and System Utilization Analysis

Mourad Bouache
Intel
AI and Performance Department
Santa Clara, CA
California, USA
bouache@yahooinc.com

Benaoumeur Senouci
NDSU
ECE Department
North Dakota State University
Fargo,USA
Ben.senouci@ndsu.edu

Abstract—**The objective of this project is to perform automated classification of anomalies of system within a large scale cloud production environment using neural network based models on time series data. In large clusters of mixed workload servers, the ability to automatically identify abnormal system utilization is a challenge due to the scale of the problem. To solve this problem, we use deep learning modeling techniques, Long-Short Term Memory (LSTM) models. The data set, to build and test the models, is from production systems over the span of two weeks. The models will use utilization metrics such as CPU, Memory, Network IO, Process Run Queue and Open Files. Anomalous usage in the production cluster is classified as (1) very low usage - less than 5% across selected metrics and (2) known anomalous behaviors like memory leaks. This paper will explain how we can create a model that will identify the anomalies we want to flag, in the real world data. We are using Intel Optimized TensorFlow in containers distributed within a cluster of TensorFlow servers.**

Keywords—System Utilization, CPU Utilization, Performance, Deep Learning, Neural Network, LSTM, Anomaly Detection, Performance Engineering.

I. Introduction

A neural network is made of an input layer, a hidden layer, and an output layer. Each layer includes multiple nodes, or neurons, and dictate the input, make inferences from those inputs in the hidden layers, and then outputs the results. The synapses are the connections between all these neurons, pretty much like the brain. If we compare the typical way a computer thinks, we give them input and then an output is generated.

This study is to understand resource utilization: CPU and Memory at Verizon Datacenters, classify hosts based on workload type and further identify systems with abnormal utilization patterns, compared to their peers within the host clusters.

The problem. In large clusters of servers we are striving for the ability to automatically identify abnormal system utilization. With clusters that span thousands upon thousands of nodes, monitoring individual servers is typically impractical. Therefore, we must apply automated, preferably autonomous, systems for classifying the cluster hosts and their workload patterns, to identify outliers and anomalies in utilization.

Using machine learning or deep learning approaches, we believe that it is possible to build a system that can identify the outliers with high degree of confidence. At the same time, account for seasonal patterns, unexpected peaks of traffic, or other atypical patterns of desirable system behavior within the sample server clusters.

In Section 2, we provide background about the deep learning platforms used in this study: TensorFlow and Intel Deep Learning solutions. In Section 3, we will talk about the LSTM model for time-series data. Experimentation environment is covered in Section 4 where we will discuss the classification and clustering process where we compare the normal to the abnormal operations during different resource usage. We will also delve into the experimentation environment in Section 5 with a distributed containers running the TensorFlow clusters. Results will be exposed within the same section. In Section 6 we conclude with some recommendations on anomaly detection using neural network and next steps.

II. Background

In this section we are presenting different technologies that we are leveraging for this work. For training we are using Tensorflow[1] on GPUs and inference using on general purpose CPU clusters using Intel Deep Learning Reference Stack built on Clear Linux with optimized Eigen, Intel MKL-DNN, and AVX512-DL Boost and VNNI for Tensorflow in containers.

A. *Deep Learning Platform: TensorFlow*

TensorFlow [1] is one of the leading deep learning and machine learning frameworks today. Earlier in 2017, Intel worked with Google to incorporate optimizations for Intel Xeon processor based platforms using Intel Math Kernel Libraries

[1] http://www.tensorflow.org

978-1-6654-5818-4/22/$31.00 ©2022 IEEE

(Intel MKL: see section 2.3). These optimizations resulted in orders of magnitude improvement in performance – up to 70x higher performance for training and up to 85x higher performance for inference.

B. *Intel Advanced Vector Extensions*

In addition, the Intel Xeon Scalable processor includes Intel Advanced Vector Extensions 512 (Intel AVX-512) [2], originally introduced with the Intel Xeon Phi processor product line. The Intel Xeon Scalable processor introduces new Intel AVX-512 CPUID flags (AVX512BW and AVX512DQ) as well as a new capability (AVX512VL) to expand the benefits of the technology. The AVX512DQ CPUID flag is focused on new additions for benefiting high-performance computing (HPC) [4] and machine learning workloads.

C. *Intel Math Kernel Library*

The optimizations discussed in this article utilize the Intel Math Kernel Library [5] for Deep Neural Networks (Intel MKL-DNN). This is an open source performance library for Deep Learning applications, intended for acceleration of DL frameworks on Intel architecture. Intel MKL-DNN includes highly vectorized and threaded building blocks for implementation of convolutional neural networks with C and C++ interfaces. Note that TensorFlow currently supports the open-sourced Intel MKL-DNN as well the DNN primitives [4] in the closed source Intel Math Kernel Library. The version to use is selected when building TensorFlow. It is expected that in the future the support for the closed source DNN primitive will be removed from TensorFlow.

III. Deep Learning Model: LSTM for Time-Series

A popular choice for this type of model is Long-Short-Term-Memory (LSTM)-based models. With sequence-dependent data, the LSTM modules can giving meaning to the sequence while remembering (or forgetting) the parts it finds important (or unimportant). Sentences, for example, are sequence-dependent since the order of the words is crucial for understanding the sentence.

Time series prediction can be generalized as a process that extracts useful information from historical records and then determines future values. Learning long-range dependencies that are embedded in time series is often an obstacle for most algorithms, whereas Long Short-Term Memory [3] (LSTM) solutions, as a specific kind of scheme in deep learning, promise to effectively overcome the problem.

A time series is a sequence of discrete data values ordered chronologically and successive equally spaced in time. In this study, the values are the performance metrics such as CPU utilization and memory, Figure 1 shows an example of a time series graph for CPU and memory utilization for a specific host. The values are defined as standard or mean deviation so the idea is to record the time and then the value will help to detect anomalies. There is a combination between the LSTM deep learning model with the times series to predict if the system, hardware interacting with the application, is entering an anomaly state or to any abnormal operation.

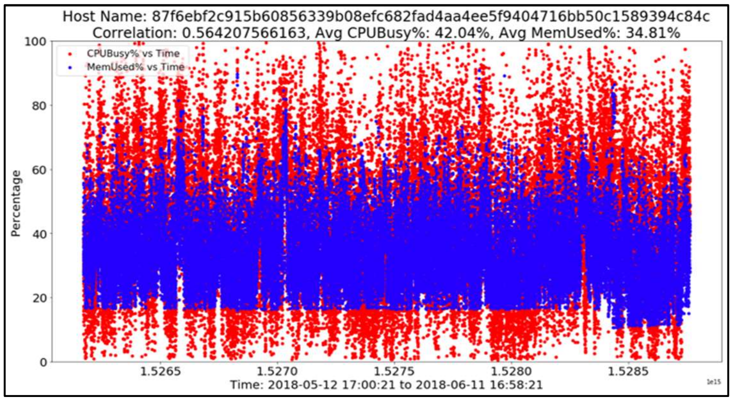


Figure 1: CPU and Memory Utilization Time series Graph

IV. Clustering and classification

In this work an anomaly is an abnormal system utilization resulting from an erroneous workload or system behavior. To identify subgroups of hosts exhibiting a particular workload patterns, we run a clustering algorithm on a set of nodes dedicated to specific compute jobs. We have observed that typically in every cluster, there is a large subgroup of nodes that is a primary workload/behavior of the cluster and a small subset of nodes that have been identified as potential anomalies. Looking at the hosts in each of the groups, the anomalous behavior was confirmed and the hosts are labeled as anomalies to be used in the training model.

Based on the system utilization graphs for the hosts in each workload we have identified the following as "normal" workloads (the color legend is below in the Appendix section):

1. ***Sporadic workloads* - see figure 2**

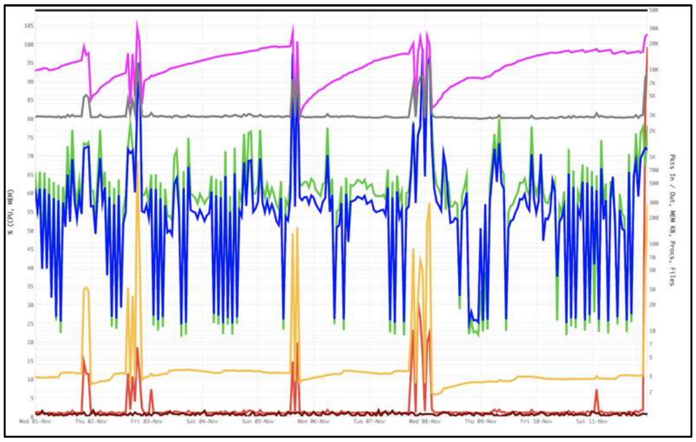


Figure 2: CPU and Memory Utilization time series representing "Sporadic" workload

2. ***Low utilization workloads - see figure 3***

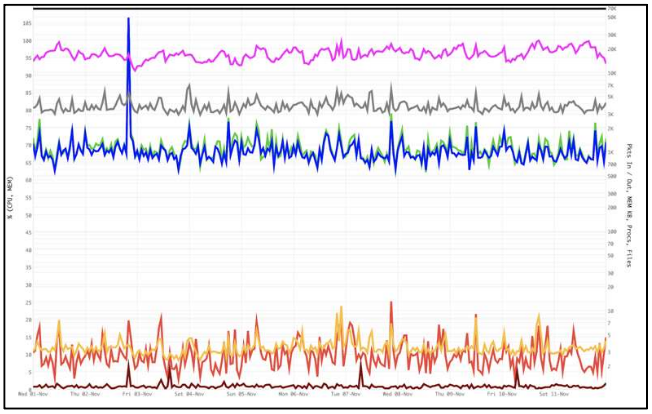


Figure 3: CPU and Memory Utilization time series representing "Low Utilization" workload

3. ***Average utilization workloads - see figure 4***

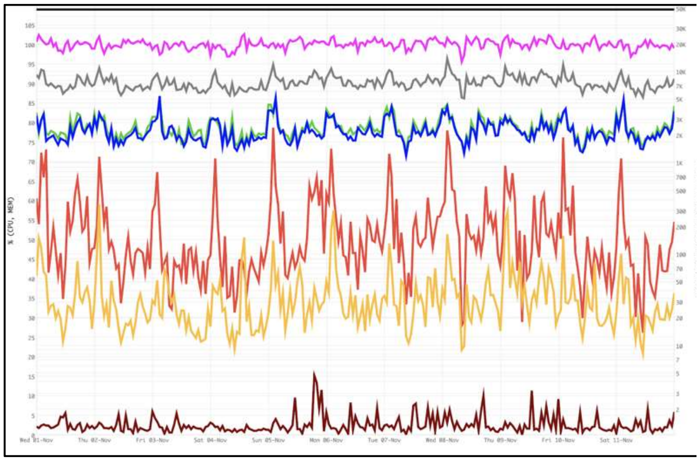


Figure 4: CPU and Memory Utilization time series representing "Average Utilization" workload

4. ***High utilization workloads - see figure 5***

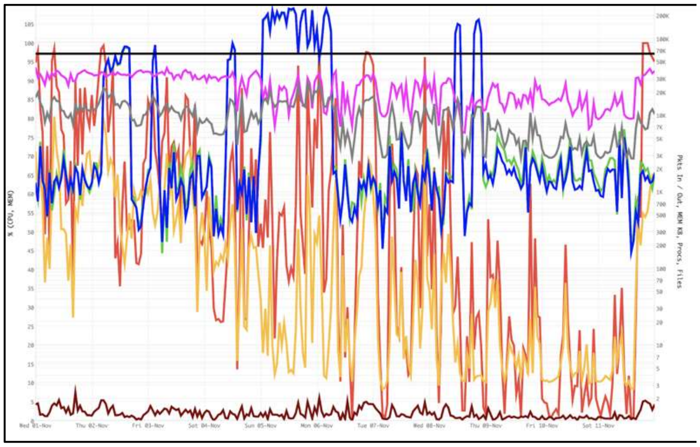


Figure 5: CPU and Memory Utilization time series representing "High Utilization" workload

We have also examined and identified the following types of abnormal workloads ("anomalies"):

5. ***Anomaly#1 - totally idle nodes (no CPU or memory utilization) - see figure 6.***

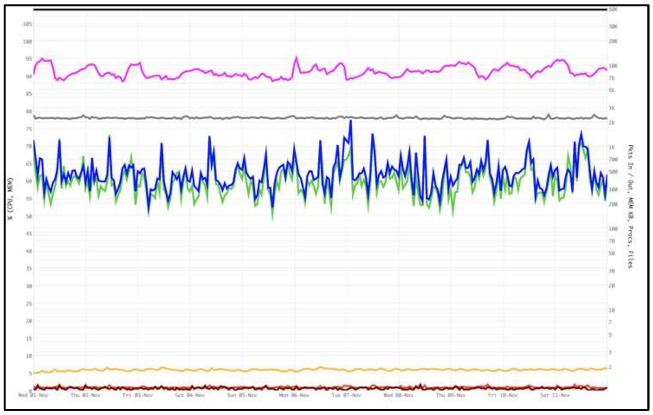


Figure 6: CPU and Memory Utilization time series representing "Anomaly#1"

6. ***Anomaly#2 - nodes that are idle, but consuming memory (there's a process, that allocated memory, yet not consuming any significant CPU) - see figure***

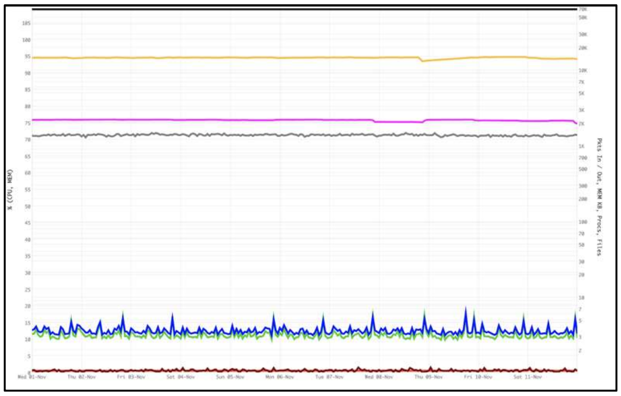


Figure 7: CPU and Memory Utilization time series representing "Anomaly#2"

7. ***Anomaly#3 - nodes that are idle, yet increasing memory consumption (there's a process that's not consuming any significant CPU, but is "leaking" memory) - see the right part of figure 8, showing increasing memory consumption.***

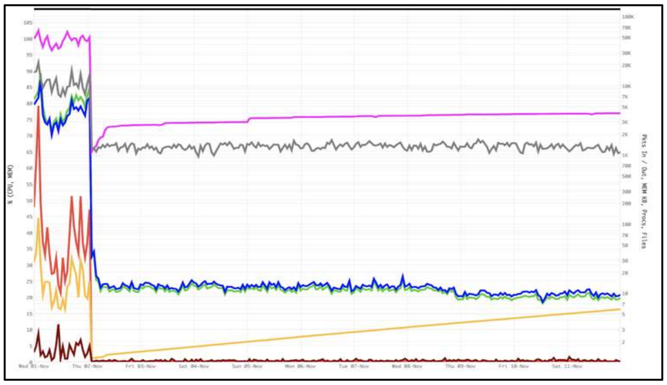


Figure 8: CPU and Memory Utilization time series representing "Anomaly#3" and "Anomaly#4"

8. ***Anomaly#4 - Utilization for these nodes shows high activity for a period of time, which ends abruptly and the node becomes idle. This indicates a node taken out of service (see the change in utilization on figure 8).***

A. *File Analysis*

The files are extracted from YAMAS database in Avro format. They contain data over the course of two weeks and are consistent across all hosts for that entire period. There are about 1000 files generated each containing 20-30 hosts with approximately 20,000-30,000 hosts sampled every minute. The metrics are sampled from standard Linux kernel registers, to capture utilization metrics across the CPU, memory, network, and disk. For the processing of the data and visualization we are using Python 3.6 with the Pandas, Numpy, and SciPy libraries and seaborne for visualization.

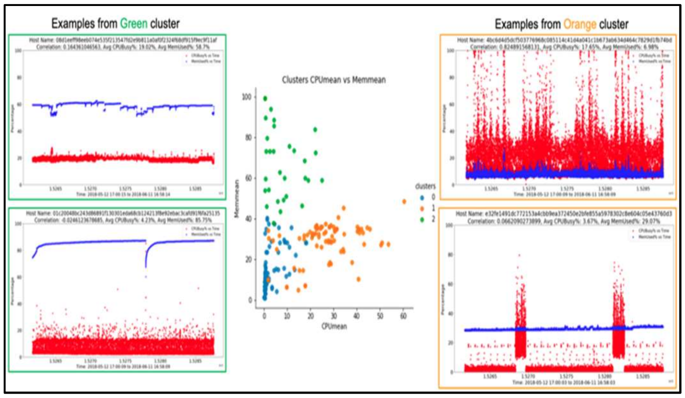


Figure 9 : File Analysis for 2 different Clusters (Green and Orange)

B. *Anomaly Detection Motivation*

The motivation for moving towards Deep Learning models started by first understanding the type of data and the general characteristics of anomalies in the data center environment. To get a general understanding of the data, the first step was to characterize the time series metrics by mean, min, max, standard deviation over the entire time window. Figure 10 shows a graph of the clusters generated from the summary statistics of the metrics *CPU Busy %* and *Active Memory* using *k*-Means algorithm with *k*=15. We find that average usage throughout the data should range between 20-50% depending on the application. This accounts for time users are inactive, for example from the hours of midnight to 4am, and general server provisioning to account for peak usage times/events. The main suspects for anomalies come with the very high memory usage and very low to no CPU usage. Of the 8000 machines in this clustering study, about 250 machines fell into cluster 8 which is the most extreme set of machines that exhibit this characteristic.

Each clustering of hosts ended up being very closely linked to specific workloads, with a few exceptions. We used this clustering to help identify which hosts were assigned to different clusters and checked to see what kind of behavior those hosts were exhibiting. It turns out those hosts were acting like the anomalous behavior described above. In the next section we will discuss how we used two of those workloads to build our LSTM models.

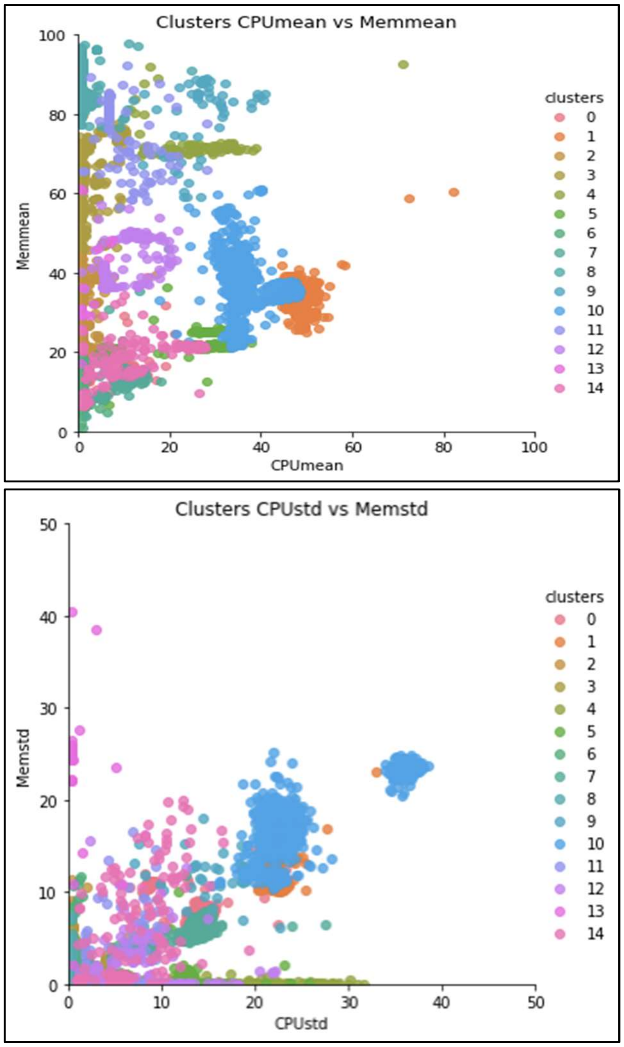


Figure 10: 8000 machine cluster based on CPU and memory min, max, mean, standard deviation using k-Means with k=15

V. Experimentation and Results

In this section we are going to talk about the software and the hardware deep learning infrastructure to train the model that we created. We will discuss the results within the same section. The training environment was performed on an Intel Skylake CPU machine with 8 NVIDIA V100 GPUs. The inference environment is an Intel CLX server with the Clear Linux Deep Learning Reference Stack[2] which is a container of Clear Linux and Tensorflow with CPU optimized eigen and AVX512.

A. *The Data Set*

Data is collected in a production environment through the use of Yamas2 which is a cloud monitoring system, Figure 12. The database contains hundreds of different metrics and for this experiment we have reduced it to a subset. The reduced subset is extracted into files and read into out pandas dataframe for analysis. The data set for training and testing the LSTM model is the time series data for each server with the assigned cluster information from the *k*-Means clustering algorithm. The initial experiments focus on a single LSTM model per workload to verify that it is possible to detect anomalies in a controlled environment. We identified two workloads which are good candidates to build an LSTM model on, we will call them Workload A and Workload B for anonymity. Workload A consists of 114 total hosts with 9 anomalies and Workload B consists of 78 total hosts with 4 anomalies. We reduced the full data set to metrics that are the most generally descriptive from the set. These metrics are CPU busy percent, memory total in kBs, memory used percent, memory active kBs, processes running, files open, IP packets, and IP packets out. Each host has 15840 measurements where each measurement represents a moment in time. The data is normalized between zero and one before building the model.

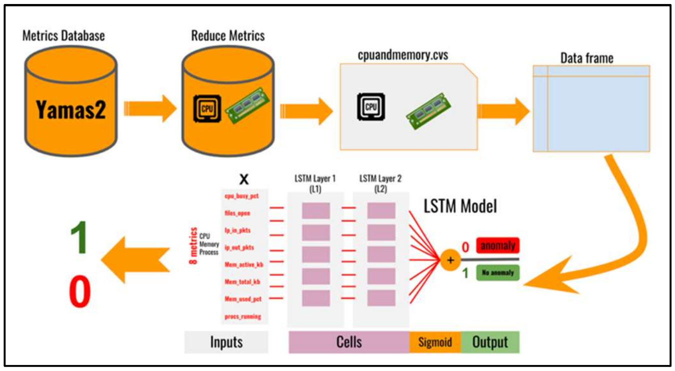


Figure 12: Data collection and manipulation flow for building LSTM models from production data

B. *Model Building*

The LSTM model was chosen due to its innate characteristics in handling time series sequential data. The final model is built with two layers where the second layer outputs to a single output which can be either an anomaly or normal behavior represented as a zero or one respectively. We choose the activation function sigmoid to keep the values bet, Figure 13.

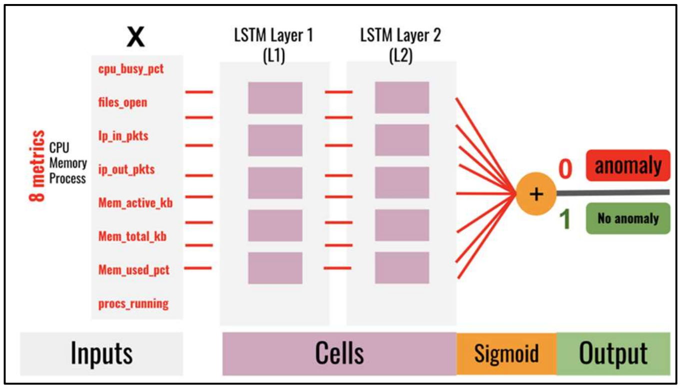
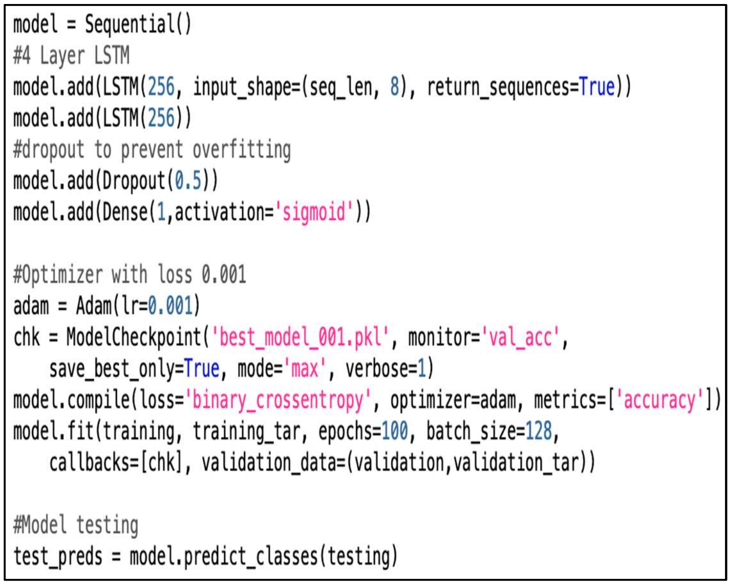


Figure 13: LSTM Model Building

The code for the model is showing in Figure 14 and written in python using keras. The model building starts with initializing a sequential model and adding LSTM layers. In this case we add two layers followed by a single dense layer since we are performing binary classification of the anomalies. The dropout of 0.5 is added to help with overfitting. We choose the Adam optimizer [6] with a loss ratio of 0.001. In future work we plan on exploring different loss ratios as well as adding more LSTM layers. The data set described in 5.a is used as the training, validation and testing data along with the binary class associated with each host represented as the *_tar vectors. The model is tested with the testing set and the accuracy is compared to the testing_tar classes.



```
model = Sequential()
#4 Layer LSTM
model.add(LSTM(256, input_shape=(seq_len, 8), return_sequences=True))
model.add(LSTM(256))
#dropout to prevent overfitting
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

#Optimizer with loss 0.001
adam = Adam(lr=0.001)
chk = ModelCheckpoint('best_model_001.pkl', monitor='val_acc',
    save_best_only=True, mode='max', verbose=1)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
model.fit(training, training_tar, epochs=100, batch_size=128,
    callbacks=[chk], validation_data=(validation,validation_tar))

#Model testing
test_preds = model.predict_classes(testing)
```

Figure 14: Python code for LSTM model

[2] https://clearlinux.org/stacks/deep-learning

C. *Results*

We used the annotated data to train the LSTM model. Approximately 50-60% of the data is used for training and the rest used for testing with an equal distribution of anomalies within each set. The results from the LSTM models using one and two layers is represented in Table 1. These first experiments were performed to test out various modeling and hyper parameter configurations. With just a single layer LSTM model, with sigmoid activation function and Adam optimization, both Workload A and B had mis-predictions for 75% of the anomalies and a few mis-predictions for the non-anomalous hosts. By adding a second layer, with the same activation function and optimizer, to the network the model accuracy went to 100%. A dropout of .5 was added to the network to help with overfitting. This showed that it is possible to classify anomalies based on utilization data generated by each host.

Table 1: Per workload anomaly detection models and their respective accuracy

	Training Set Size (#hosts)	Validation Set Size (#hosts)	Testing Set Size (#hosts)	1 Layer Accuracy	2 Layer Accuracy
Workload A	63	25	26	86.4%	100%
Workload B	45	16	17	84.3%	100%
Mixed Workload	1244	622	626	N/A	100%

The next experiment was to combining multiple workloads into a single data set and see if it is possible to classify the anomalies. This set consists of five different workloads spanning over 2000 machines. Using the LSTM model architecture as above, only changing the batch size to 32 from 128, we tried two different hyperparameter loss ratio values for the Adam optimizer of 0.001 and 0.01. The result was a testing accuracy of 100% and 100%. It seems that even with mixed workload types, each represented in Section 4, that the LSTM model performs with extremely good accuracy. Changing the loss accuracy of the optimizer had no impact on the prediction results.

VI. Conclusions and Future Works

In this study we created workload specific anomaly detection models that detect anomalous usage from within large scale production data centers. We used annotated data of two distinct workloads to train the LSTM model from two weeks of data. For each individual workload we got very good accuracy, 100%, with a two layer LSTM with a sigmoid activation layer. When combining the workloads into a single data set, the accuracy went down to 94%, but predicted all anomalies as false negatives.

The work has spawned multiple next steps that we will explore. The first thing we are going to explore the reasoning behind the accuracy levels and ensure that overfitting did not occur in the models. We would also like to explore other sequential and time series classifiers, for example Transformer. In order to put this solution into production to monitor for anomalies on a daily cadence, the models will need to be rebuilt from daily host data with tagged anomalies instead of monthly host data. The models will continue to be built on GPU systems, but the nightly inference will be performed on a general purpose containerized CPU environment. The container will be highly optimized for inference on CPUs using the Clear Linux Deep Learning Reference Stack for Tensorflow. This will run as a best effort job on the general purpose cluster and anomalies will be reported back through the alerting system.

In conclusion, we believe that deep learning models will be a great method for classifying anomalous usage in large scale production data center.

References

[1] Pattanayak, Santanu. "Introduction to Deep-Learning Concepts and TensorFlow." Pro Deep Learning with TensorFlow, 2017, pp. 89–152., doi:10.1007/978-1-4842-3096-1_2.

[2] Kusswurm, Daniel. "Advanced Vector Extensions (AVX)." Modern X86 Assembly Language Programming, 2014, pp. 327–349., doi:10.1007/978-1-4842-0064-3_12.

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." Neural Computation, vol. 9, no. 8, 1997, pp. 1735–1780., doi:10.1162/neco.1997.9.8.1735.

[4] Arora, Ritu. "An Introduction to Big Data, High Performance Computing, High-Throughput Computing, and Hadoop." Conquering Big Data with High Performance Computing, 2016, pp. 1–12., doi:10.1007/978-3-319-33742-5_1.

[5] Kalinkin, A., et al. "Intel® Math Kernel Library Parallel Direct Sparse Solver for Clusters." EAGE Workshop on High Performance Computing for Upstream, 2014, doi:10.3997/2214-4609.20141926.

[6] Zhang, Zijun. "Improved Adam Optimizer for Deep Neural Networks." 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018, doi:10.1109/iwqos.2018.8624183.

Appendix

A. *LEGEND TO THE UTILIZATION GRAPHS*

Figure #2 to #8 contain utilization graphs - see color legend below:

`cpu.busy.pct`	🟥
`ip.in.pkts`	🟩
`ip.out.pkts`	🟦
`mem.active.kb`	🟪
`mem.total.kb`	⬛
`mem.used.pct`	🟧
`procs.running`	🟫
`sys.files.open`	▨