Anomaly Detection on the Edge

Joseph Schneible, Alex Lu
Technica Corporation
Dulles, VA 20166
jschneible@technicacorp.com, alu@technicacorp.com

***Abstract*— Anomaly detection is the process of identifying unusual signals in a set of observations. This is a vital task in a variety of fields including cybersecurity and the battlefield. In many scenarios, observations are gathered from a set of distributed mobile or small form factor devices. Traditionally, the observations are sent to centralized servers where large-scale systems perform analytics on the data gathered from all devices. However, the compute capability of these small form factor devices is ever increasing with annual improvements to hardware. A new model, known as edge computing, takes advantage of this compute capability and performs local analytics on the distributed devices. This paper presents an approach to anomaly detection that uses autoencoders, specialized deep learning neural networks, deployed on each edge device, to perform analytics and identify anomalous observations in a distributed fashion. Simultaneously, the autoencoders learn from the new observations in order to identify new trends. A centralized server aggregates the updated models and distributes them back to the edge devices when a connection is available. This architecture reduces the bandwidth and connectivity requirements between the edge devices and the central server as only the autoencoder model and anomalous observations must be sent to the central servers, rather than all observation data.**

Keywords— Anomaly Detection, Autoencoder, Deep Learning, Edge Computing, Federated Learning

I. Introduction

The Internet of Things (IoT) model [1] connects many mobile and small form factor devices with centralized servers, sometimes called the cloud. Large-scale systems are required for the servers to perform analytics on potentially petabytes of data, provide real-time monitoring and actions, and gather feedback for decision-making. However, the compute capability of small form factor devices is ever increasing with annual improvements to hardware. This has led to a new branch of the IoT model, known as edge computing [2] which performs local analytics on the distributed devices. These so-called edge devices communicate with the servers only when necessary. In the old model, data is sent from the edge devices to the centralized servers for processing and analytics. In edge computing, processing and analytics occur directly on the edge device with limited communication between the edge devices and the centralized servers.

Anomaly detection [3] is the process of identifying unusual signals or observations within a larger data set and is an important task in many different fields from cybersecurity to the battlefield. The goal of anomaly detection is not only to identify anomalous observations correctly, but also to minimize instances false positives by rapidly adjusting to the latest trends in the observed data. Using the traditional IoT model, the edge devices would send all gathered data to the servers where all processing would occur and action would be taken. This requires a constant high bandwidth connection to the centralized servers and introduces additional latency into the process.

Technica's approach uses autoencoders, a specialized deep learning neural network, on each edge device to identify anomalies and adjust the model by learning from new observations. This allows the system to go beyond simple threshold measurements and identify instances of anomalous behavior between correlated variables. When a connection to the centralized servers is available, the updated models from each sensor are aggregated and distributed back to the edge devices. In this way, local models are able to learn from data they did not observe without the burden of communicating all sensor data to the central servers. This technique is called federated learning [4].

This paper describes Technica's approach to performing anomaly detection on the edge. Section 2 discusses related work. Section 3 introduces the concepts of deep learning and autoencoders. Section 4 presents the concept of federated learning. Section 5 provides details about our model. Section 6 shows our experimental results. Finally, Section 7 concludes the paper with a discussion of the experimental results.

II. Related Work

Anomaly detection is a widely studied problem with a broad range of applications and a diverse set of approaches including machine learning and statistical approaches [3] [9]. Applications include network intrusion detection, fraud detection and identification of business trends in e-commerce.The use of auto-encoders as a state-of-the-art method for anomaly detection [5] [6] [7] is well documented.

The concept of federated learning was proposed in a Google research paper discussing its application to deep convolutional networks for the use of language modeling and image classification tasks on mobile smartphone devices [4]. While the general architecture is similar to what we present in this paper, we apply federated learning to the anomaly detection task in an edge computing environment.

III. Deep Learning

Deep Learning [8] is a relatively new area of machine learning. Based loosely on the human brain, deep learning

978-1-5386-0595-0/17/$31.00 ©2017 IEEE

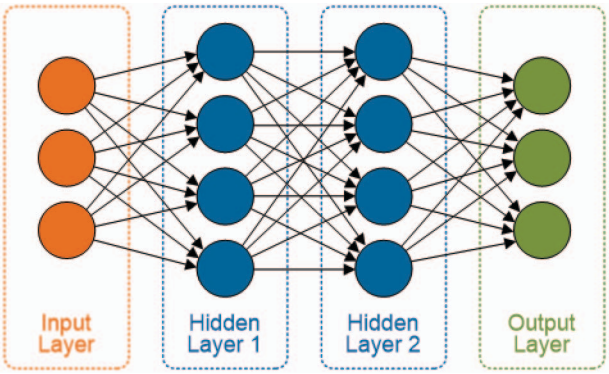

Fig. 1 An artificial neural network with two hidden layers and all layers fully connected

models are comprised of multiple processing layers and learn representations of the data at multiple levels of abstraction. Deep learning has been used in a wide variety of applications, such as speech recognition, language modeling and image analysis. It has produced state-of-the-art results in many fields.

A. *Artificial Neural Networks*

A deep learning model, also called an artificial neural network (ANN), takes as input a vector of values, performs a set of simple calculations defined by the structure of the ANN and produces an output. The input values could represent the pixels in an image, words in a sentence, or a set of heterogeneous features and the output could represent a label. A process called training takes known input and the expected outputs, and modifies the network based on the difference between the expected and actual output of the neural network to better reproduce the expected results. The network then attempts to produce an output that matches the expected behavior, such as correctly identifying the breed of a dog in an image.

Between the input and output layers are one or more hidden layers, as seen in Fig. 1. The multiple hidden layers give deep learning its name. These hidden layers are comprised of neurons that perform very simple computations on their inputs and pass the result to neurons in the next layer. The structure of the connections from one layer to the next can vary from layer to layer and the weights on the connections determine how strongly a given neuron in one layer affects the input of a neuron in the next layer.

During the training process, a set of inputs with corresponding expected outputs are fed into the network. Errors found by comparing the output of the network with the correct outputs are used to make adjustments to the weights of the connections between neural network layers. The training process is often performed by using the stochastic gradient descent technique. Each layer will learn conceptually more complex features of the data. For example, the early layers of a network trained for facial recognition will learn simple characteristics like lines or curves, while later layers will learn body parts and faces.

B. *Autoencoders*

Autoassociative neural network encoders, or autoencoders for short, are a specific type of neural network that attempt to reproduce the input data as their output. They are comprised of two symmetric neural networks, as seen in Fig. 2. The first network encodes the input data into a compressed form and the second decodes the compressed form to approximate the input. The middlemost layer learns a dimensionally reduced representation of the input data. This reduced representation will reproduce common inputs well, while uncommon inputs will see more significant errors in reproduction. This error in reproduction is the basis of the anomaly score which determines whether an observation is anomalous or not.

IV. Federated Learning

Using the traditional approach, training a neural network requires having a single copy of the model and all of the training data in one place. In many real world scenarios, data is gathered across an array of sensors. In those scenarios, all sensor data would have to be sent to a central server for training and the resulting network weights distributed back to the sensors. However, these sensors often have limited bandwidth and intermittent connections to the central server. Federated learning allows the model to be trained on each edge device and is based on the data parallelism model.

In the data parallelism model, multiple copies of the neural network are created. The training data is split between the copies of the network, such that each copy is trained on an independent section of data. Once all copies of the model are trained, the resulting weights are aggregated at a central repository. This is usually accomplished by averaging the weights of the independently trained models. It is easy to see how this translates to edge computing, where each edge device trains a copy of the neural network using the data that it observes. The new network weights are sent to the central server for aggregation, and the resulting model is then distributed to the edge devices. In this way, the model at each edge device will have learned from the data gathered from all devices without having to transmit the full training to the central server. This greatly reduces bandwidth requirements and can occur when a connection to the central server is available.

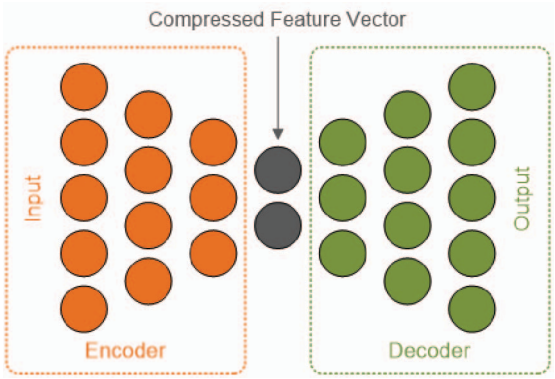

Fig. 2 An autoencoder network

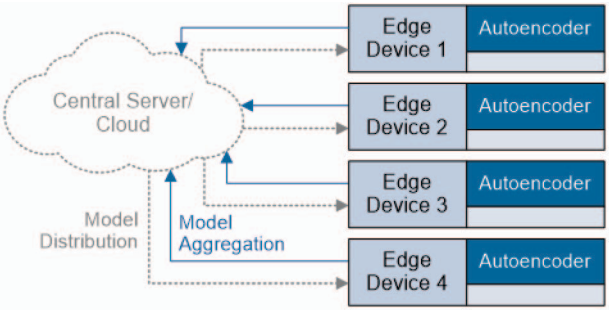

Fig. 3 Architecture of anomaly detection on the edge

V. Anomaly Detection on the Edge

Technica's anomaly detection on the edge model is comprised of two elements, as seen in Fig. 3. One is a distributed set of devices each deployed with a copy of the autoencoder and the other is a central server for aggregating and distributing changes to the trained autoencoder network weights. Multiple rounds of training and aggregation may occur in order to produce better results.

The edge devices independently gather observations and determine an anomaly score for each. As seen in (1), the anomaly score, A, is determined by the Euclidean norm of the difference between the input and output of the autoencoder where *i* is the vector of inputs and *o* is the vector of outputs. A threshold is specified to determine at what point an observation is considered anomalous.

$$A = \sqrt{\sum_j |o_j - i_j|^2} \quad (1)$$

The number of layers in the network, the number of neurons in each layer, the functions performed by each neuron, the optimizer for training and the loss function are all allowed to vary depending on the data set and use case.

Concurrent with the anomaly detection process, each edge device trains the local autoencoder with observations that are determined to be non-anomalous. In this way, the autoencoder adjusts to the latest trends in the data to limit the number of false positives.

VI. Experiments

This section details experiments conducted to show the viability of working with an autoencoder in a distributed environment. Experiments 1 through 4 were performed on the classic shuttle data set as preprocessed in [10]. The data set contains 46,462 data points with 9 attributes. Approximately 1.89% of the data points are anomalous. In CSV format, this data set is 1.2MB in size. Experiment 5 was performed on the network intrusion detection data set from the KDD 1999 Cup. This data set contains 620098 data points with 29 attributes. Approximately 0.17% of the data points are anomalous. In CSV format, this data set is 75MB. Both data sets are available online [11].

The autoencoders used in these experiments were comprised of five hidden layers with 64, 32, 16, 32 and 64 neurons with ReLU activation functions, respectively [12]. The input and output layers have a number of neurons equal to the number of attributes in the data set. The weights of the trained models take up 72KB of memory for the shuttle data set and 102KB for the KDD99 data set.

A. Experiment 1

In Experiment 1, we sought to prove the effectiveness of merging models via averaging their weights. Using the shuttle dataset, we simply split training data into three equal portions to train three separate models. Each model was trained using the same number of epochs, batch size and autoencoder architecture / parameters. To test the effectiveness, we used a test set of data with both normal and anomalous data. Each model was evaluated using the F1 score based on the autoencoder's ability to label test instances.

Table I. Results from Experiment 1

Model Name	Training Data Samples	False +	False -	F1
Edge 1	13900	1	36	0.9785
Edge 2	13593	1	37	0.9779
Edge 3	13215	1	36	0.9785
Merged Model	40708	0	0	1
Full Model	40708	0	38	0.9779

In Table I, the performance of the merged model, i.e. the model produced by averaging the weights of Edge 1 through 3, results in perfect classification in our testing data. Surprisingly, this outperforms the model trained on the full set of data which only achieves an F1 score of 0.97788. This experiment shows that the merged model can perform as well as, if not better than, a model trained on all of the data at once. The total bandwidth required to merge the models and redistribute them is 432KB, approximately a third of the original data set size.

B. Experiment 2

In Experiment 2, we sought to show how the federate autoencoders performed when the definition of normalcy changed over time. For this experiment, a univariate dataset was generated with a linearly separable boundary from normal and anomalous data. Two sets of normal data were generated, one with random values bounding within a range and another with random values bounding within a new range (with some overlap). Both of these training datasets were linearly separable from the anomalous data.

Table II. Results from Experiment 2

Model Name	Training Samples	False +	False -	F1
Base Model	643	408	0	0.2273
Edge 1	400	441	0	0.2139
Edge 2	400	0	0	1
Edge 3	400	0	0	1
Merged Model	1843	0	0	1
Full Model	1843	0	0	1

We initially trained a base model using the original training data, which would theoretically be deployed to each edge device. To simulate a change in normalcy, we used a new set of data as the testing criteria to evaluate the classification. As expected, the false positive rate jumped as the base model was unfamiliar with the shifted definition of normalcy. The false negative rate remained at zero. To improve the base model, we trained using the new ranged data and retested the false positive rate. To improve one of the models further, we merged the models via averaging the weights to produce a final merged model. This model effectively classified all normal and anomalous instances.

With a generated data set like this, it is not surprising that the merged and full models performed so well. However, the comparison to the base model shows that a case where the data trend changes over time, the autoencoder can constantly learn the new trends and make updates to the local model. These local models can eventually be sent and aggregated to produce a new consensus model.

C. *Experiment 3*

In Experiment 3, we compared the performance of our federated model trained on data that is not independent and identically distributed (IID) to a single model trained on the full training data set. To create a non-IID data set, we applied procedures similar to that used in [4] to the shuttle data set. We sorted the non-anomalous data and divided it into twice as many segments as we had edge models to be trained. To generate data to train each edge model, we combined two segments of the non-anomalous data that were evenly spaced half the distance of the total number of segments along with a proportionate number of anomalous data points. 10% of the data points were reserved for testing.

In this scenario, merging the federated models after fully training them produced poor results. Instead, we applied the federated learning technique from [4] and randomly selected a fraction of the models to be partially trained, averaged and redistributed to the edge. This process was repeated with a different randomly selected fraction of the models over multiple rounds of aggregation. The federated models were trained for 2 epochs between each iteration of aggregation and merged 100 times and the client fraction was held constant at 0.4.

The threshold for the anomaly metric in each test was determined by the threshold value which most closely reproduced the anomaly rate when inferencing on the training data. This assumes that the user knows the approximate anomaly rate, but nothing else about the data set. All federated models were trained with 100 iterations of 2 epochs each. The full model was trained for 100 epochs, at which point it had converged.

Table III. Results from Experiment 3

Model Type	Number of Federated Models	Client Fraction	F1
Federated	5	0.4	0.9695
Federated	10	0.4	0.9545
Federated	20	0.4	0.9655
Full	n/a	n/a	0.9605

As seen in Table III, the federated model performed approximately as well as a single model trained on the full training data set. In some instances, the federated model actually outperformed the full model.

D. *Experiment 4*

In Experiment 4, we compared the performance of our federated model trained on the KDD99 data set for various numbers of federated learning iterations. As in the previous experiment, the data set was sorted and 10% was reserved for testing. The results in Table IV show that 100 iterations are sufficient to achieve good results on this larger data set. The full model was trained for 300 epochs. This was sufficient for convergence.

Table IV. Results from Experiment 4

Model Type	Iterations	False +	False -	F1
Federated	100	11	4	0.9315
Federated	200	8	4	0.9444
Federated	300	7	8	0.9289
Full	300	0	2	0.9905

VII. CONCLUSION

Our experimental results, while preliminary, show that a distributed approach to anomaly detection using autoencoders can produce similar results to a non-distributed model. The distribution allows most of the computation to occur in parallel on the edge devices. Additionally, it can reduce the amount of data required to be transferred to the central server making it viable for situations when the edge devices have limited or inconsistent connections. As seen in the second experiment where the model trained on the first edge device produced poor results, the overall distributed model was robust to these errors.

In instances where the data is not independent and identically distributed, the models are merged over multiple iterations of aggregation. Experiment 3 showed that the federated models can perform as well as, and in some instances, better than the full model. On the larger data set in

Experiment 4, the federated model performed well, but not as well as the full model.

There is a possibility that different datasets will require a large amount of variation in the architecture of the autoencoder network. Our initial tests have not required any variation in our original architecture; however, we concede to the possibility. Most of the optimization work has been in regards to modifying the deep layers' size and activation functions, selecting and configuring an optimizer for training and selecting a loss function. These processes would require someone with working knowledge of neural networks to best optimize these parameters to guarantee success with training an autoencoder.

REFERENCES

[1] H. Kopetz, "Internet of Things," in *Real-Time Systems*, 2011, pp. 307-323.

[2] H. Pang and K. Tan, "Authenticating Query Results in Edge Computing," in *IEEE International Conference on Data Engineering*, 2004.

[3] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys,* vol. 41, no. 3, 2009.

[4] H. B. McMahan, E. Moore, D. Ramage, S. Hampson and B. Aguera y Arcas, "Communication-Efficient Learning on Deep Networks from Decentralized Data," *AISTATS,* 2017.

[5] M. Martinelli, E. Tronci, G. Dipoppa and C. Balducelli, "Electric Power Systems Anomaly Detection Using Neural Networks," in *International Conference on Knowledge-Based and Intelligent Information and Enginneering Systems*, 2014.

[6] H. Ullah, M. Ullah and N. Conci, "Real-time anomaly detection in dense crowded scenes," *Video Surveillance and Transportation Imaging Applications,* 2014.

[7] A. Dau, V. Ciesielski and A. Song, "Anomaly detection using replicator neural networks trained on examples of one class," in *SEAL*, 2014.

[8] Y. LeCun, Y. Bengio and G. Hinton, "Deep Learning," *Nature,* 2015.

[9] M. Agyemang, K. Barker and R. Alhajj, "A comprehensive survey of numeric and symbolic outlier mining techniques," *Intelligent Data Analysis,* vol. 10, no. 6, pp. 521-538, 2006.

[10] M. Goldstein, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," *PLOS ONE,* 2016.

[11] M. Goldstein, "Unsupervised Anomaly Detection Dataverse," 2015. [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/OPQMVF. [Accessed 25 July 2017].

[12] F. Chollet, "Building Autoencoders in Keras," Keras, 14 May 2016. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html. [Accessed 28 July 2017].