DeepSecurity
Cybersecurity Threat Behavior Classification

Isaac J. Faber
Mangement Science and Engineering Stanford University
Email: ifaber@stanford.edu

Giovanni S. P. Malloy
Mangement Science and Engineering Stanford University
Email: malloyg@stanford.edu

***Abstract*—Identification of network traffic originating from threats is an ongoing challenge in cybersecurity. Servers continuously receive requests from all over the world. Identifying the proper subset of traffic requests as adversarial is essential. Our DeepSecurity model uses a feed-forward neural network to classify network traffic as either a threat or non-threat.**

I. Introduction

We explore the topic of threat identification in cybersecurity. Cyber-attacks are a set of discrete, observable steps called a 'kill chain'. When defending a computer system, it is typical for a significant amount of data to be produced from security devices. Appliances like firewalls, intrusion-detection-systems, and proxies generate data useful for analysis. However, the volume of security-related data is such that comprehensive human evaluation is impractical. Much of current practice is focused on known threat signatures which inform the defender's security posture. Security posture can be thought of as a set of 'gates' which have some probability of legitimate or threat traffic flowing through them.

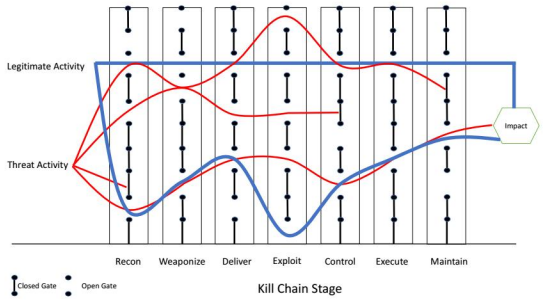

Fig. 1. Network Defenders Control a Set of Gates

The defender's job is to find the optimal gate policy that allows for minimal threat activity while allowing as much legitimate activity as possible. Many threats (such as hackers, malware, etc.) have specific non-linear behavior which may manifest within data collected by network defenders. The difficulty is knowing which gates to keep closed and which to keep open. The number of gates is enormous and to difficult to evaluate manually. For example, gates include:

- IP Address: Source and Destination
- Port: Source and Destination
- Time Stamp
- Packet content

However, using deep learning techniques, we can help differentiate between legitimate network traffic and bad behavior. Using a machine to make low-level gate decisions means that human analysts will only see signals of sufficient importance. This classification task can improve the efficiency of security personnel and automate many of the currently manual processes.

II. Methods

A. Data

Security data is considered sensitive by most organizations, so any methods developed would need to be generalized into similar (but not identical) environments. Also, there has not been much work done in the area of deep learning applied to network level threat detection in this way so most techniques will be new, untested and not useful for transfer learning. One of the team members (Isaac) has personally collected a unique dataset using honeypots and categorized threats using an open source blacklist. There are several open source datasets such as the publicly provided DARPA / MIT challenge data (https://www.ll.mit.edu/ideval/data/). For this project, the blacklist data contains IP addresses that are known to conduct illegitimate network traffic. By cross-referencing the honeypot IP addresses with the blacklist, we were able to label the honeypot data for training purposes as either a threat or non-threat.

We currently have over 600,000 network events in our dataset. Therefore, to determine train, dev, and test sets, we will first randomly sort these network events. Then, from the randomized order, we will select the first 540,000 to be in the training set, the second 30,000 to be in the dev set, and the last 30,000 to be in the training set. These datasets will all come from the same data distribution.

B. Features

As discussed in the introduction the amount of gates is extremely large (contain all IP address, ports, and combinations). In order to build a manageable feature set, this project makes use of custom developed encodings (similar to NLP modeling) using expert input. The input layer of the model X will include three categories of encoded features: time encoding, port encoding, and IP encoding. These encodings come from the 600,000 network events of the honeypot data

and are done by unique IP address. The time encoding includes 2 parameters:

- the average of the hour of the timestamp (GMT)
- the standard deviation of the hour of the timestamp (GMT)

The average hour of the day, t_i will be a discrete input from 1 to 24 corresponding to the i^{th} hour of the day in Greenwich Mean Time.

The next subset of parameters comes from the port encoding of the data. There are 6 port encoding parameters. The first is a binary variable indicating whether the protocol request is regular (1) or irregular (0). The next input parameter is the source port of the network event. The following three port-related parameters are a one-hot vector used to classify the destination and source port:

- common database ports
- standard network ports (i.e. 22, 80, 443, etc.)
- the number of unique ports

Finally, the IP encoding includes 3 input parameters:

- total frequency of requests
- Geo-code: latitude
- Geo-code: longitude

Using these encodings avoids the pitfalls of large one hot vectors. For example, our dataset contains 473 destination ports. Using a one-hot vector to represent all of those would result in 472 parameters with value 0 (the destination ports not asked by the training example), and 1 with value 1 (the destination port requested by the training example). A large number of zeros input into the neural network will result in a sparse network. Therefore, by using creative encodings, we avoid a sparse network that is unable to learn.

C. Network Architecture

We use a feedforward network to implement DeepSecurity. The network in its original form (Figure 2) had three layers and a softmax output. The hidden layers are linear with ReLU activation functions. This is a generic network architecture that allows us to build an initial implementation of the model quickly and focus more on iterations to improve performance. Since we want the output to classify as 0 (not a threat) or 1 (threat), the softmax function with two outputs is the best for the last activation function. In our fully connected network, we have 478 corresponding neurons per hidden layer and 1 nueron for the activation layer. The resulting size of the parameters of our model is as follows: $W^{[1]} = [478, 9], b^{[1]} = [478, 1], W^{[2]} = [478, 478], b^{[2]} = [478, 1], W^{[3]} = [478, 2], b^{[3]} = [2, 1]$. This means that the model will handle 458,403 different parameters. This is a manageable number of parameters and is easily reducible with changes in network architecture as needed. However, using this network architecture still produced both bias and variance.

We used three additional network architectures to address the issues of bias and variance. To reduce bias in our model, we used a bigger network (Figure 3).

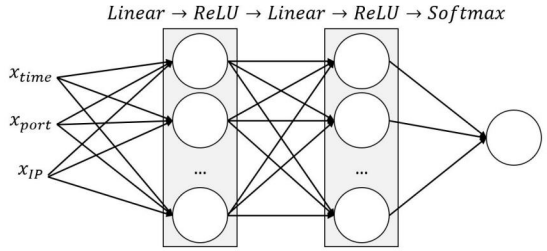

Fig. 2. The network architecture involves two hidden layers that are linear with ReLU activation functions and a sigmoid activation layer.

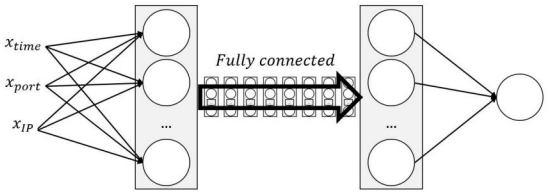

Fig. 3. The network architecture involves ten hidden layers that are linear with ReLU activation functions and a sigmoid activation layer.

The bigger network contains 11 fully-connected layers instead of 3 which will avoid underfitting the model. The resulting size of the parameters of the bigger network are as follows: $W^{[1]} = [12, 9], b^{[1]} = [12, 1], W^{[2]} = [12, 12], b^{[2]} = [12, 1], W^{[3]} = [12, 12], b^{[3]} = [12, 1], W^{[4]} = [12, 12], b^{[4]} = [12, 1], W^{[5]} = [12, 12], b^{[5]} = [12, 1], W^{[6]} = [12, 12], b^{[6]} = [12, 1], W^{[7]} = [12, 12], b^{[7]} = [12, 1], W^{[8]} = [12, 12], b^{[8]} = [12, 1], W^{[9]} = [12, 12], b^{[9]} = [12, 1], W^{[10]} = [12, 12], b^{[10]} = [12, 1], W^{[11]} = [2, 12], b^{[11]} = [2, 1]$. Much like the original network, each of these layers is feed-forward. Yet, the number of connecitons between layers is much smaller in order to reduce computation time. Therefore, the total number of variables is only 1,550.

To reduce variance in our model, we used regularization in the form of dropout(Figure 4). We chose to use dropout because of our relatively large layer sizes. Dropout effectively shrinks the network and spreads out the weights through the model. We first implemented the dropout model on the 3 layer network to determine the effects of dropout independent of the effects of a bigger network. The amount of regularization effect from dropout is dependent on the probability of keeping a neuron in the network. As the probability of keeping a neuron increases, the regularizing effect decreases. However, as the probability of keeping a neuron in the network goes up, the amount of training set error decreases. To balance the benefits of regularization with the adverse effects of training set error, the probability of keeping a neuron in our dropout model is 0.5.

Finally, we combined the 11 layer network with dropout in order to combine the benefits of reduced bias and reduced variance (Figure 5). We expect that the benefits will be subadditive because the larger network will increase variance and the regularizing effect of dropout will increase bias.

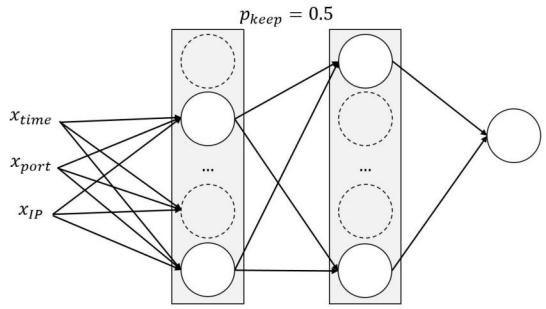

Fig. 4. The network architecture involves two hidden layers that are linear with ReLU activation functions and a sigmoid activation layer with dropout implemented in each hidden layer (keep probability = 0.5).

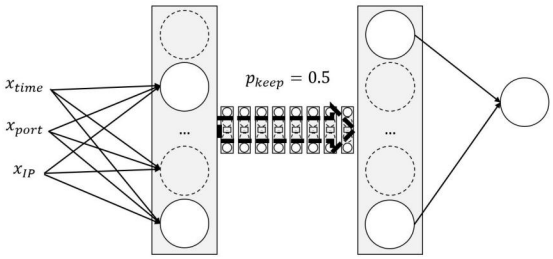

Fig. 5. The network architecture involves ten hidden layers that are linear with ReLU activation functions and a sigmoid activation layer with dropout implemented in each hidden layer (keep probability = 0.5).

D. Loss Function

We use a standard cross-entropy loss function to run DeepSecurity: $\mathcal{L}\{y, \hat{y}\} = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$. This loss function is ideal for the threat/non-threat binary classification of this model. Therefore, the cost function for this model is $\mathcal{J} = \frac{1}{m}\sum \mathcal{L}\{y, \hat{y}\}$. We will use an AdamOptimizer to minimize costs over the cross entropy loss function. The AdamOptimizer will use the standard hyperparamter values of: $\beta_1 = 0.9, \beta_2 = 0.999, and \epsilon = 10^{-8}$. The AdamOptimizer is the best choice for our problem because it is easy to configure with only the hyperparamter value, α, needing training. It combines the advantages of gradient descent and RMSProp and is best equipped to handle sparse gradients.

E. Hyperparameters

The choice of hyperparameters - learning rate, epochs, and minibatch size - was crucially important, as hyperparameters affect all subsequent parameters. After testing over a wide range of values, the best values for this early application where

- Learning rate $= 0.001$
- Epochs=1500
- Minibatch Size = 32

This resulted in the best model performance. In testing these values, we examined learning rates from 0.1 to 0.00001 and Epochs from 200 to 2000. We also varied the minibatch size extensively in multiples of 2 from 32 to 1,024. Over the experiments, we found that a minibatch size of 32 performed the best and did not significantly impact run time of the model.

F. Measuring Error

To measure performance, we will look to quantify error. By comparing our results to the blacklist dataset, we can determine the error of the training set, development set, and the test set. Using these measurements of error, we will be able to decide on the levels of variance and bias from our model. For cybersecurity applications, false positives are especially resource-intensive errors. Therefore, we will focus on minimizing false positive results to improve precision and recall related to thresholds for alerting a human analyst. Since we are concerned with false positives, to evaluate DeepSecurity, we will be more concerned with precision than recall or F1 score to measure error. However, we will measure accuracy, precision, recall (senstivity), F1 score, and specificity for each model variant.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

III. Results

A. Performance Metrics

The model was run using mini-batch gradient descent with and Adam optimizer using TensorFlow in python. Table I shows the variability among different network architectures. Overall, deep learning methods outperformed traditional machine learning methods, like logistic regression, in all measures. Accuracy was highest with the original 3 layer network, the enlarged 11 layer network, and the 11 layer network with dropout. Precision was the weakest performance measure for all of the different model structures. Precision is the proportion of threat detections that are true threats rather than false postivies. This indicates that more than half of the threats detected by the model are not actually threats. The 11 layer network performs the best of any network in precision. Recall and F1 score are less important measures for this application, as we aim to minimize false positives. Recall representa the proportion of actual threats that were detected. In this case, all of the network architectures performed relatively well. As expected, the F1 scores fell inbetween the precision and recall scores for all network architectures.

TABLE I
Test Model Results

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.59	0.40	0.59	0.46
3 Layers	0.64	0.44	0.64	0.51
Dropout	0.63	0.42	0.63	0.49
11 Layers	0.64	0.47	0.62	0.53
11 Layer (Dropout)	0.64	0.45	0.64	0.53

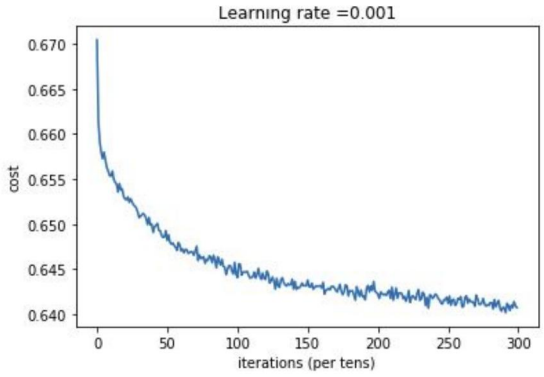

Fig. 6. The cost function of the three layer network over 1500 epochs.

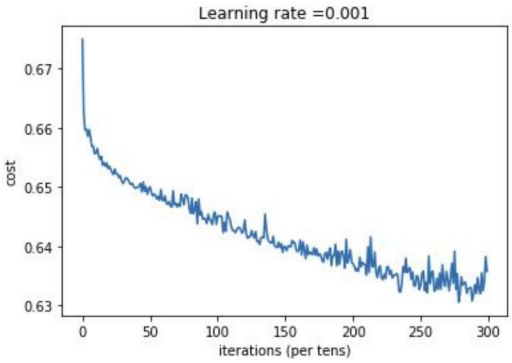

Fig. 7. The cost function of the eleven layer network over 1500 epochs.

B. Costs

The cost improvement for all models is generally consistent. The costs improve ove epochs but level out quickly with only marginal improvement after epoch 100. As expected with minibatch implementation, the cost experiences some perturbations over time. These are minor when compared to the general trend of the model costs. These perturbations are due to the stochastic nature of a minibatch implementation. The original network architecture and the larger network architecture cost improvements (Figures 6 and 7) are not dramatic, but they do show measurable improvement over time. This indicates that the model is learning from the dataset, albeit slowly. As the network depth increases, the overall cost computed also decreases. This reflects the overall improved performance as the network grows. The dropout network did not improve costs after the first iteration (Figure 6). This is likely because our model did not have much variance. Therefore, regularization did not improve the costs associated with the model.

IV. Conclusions

The initial performance of these models against the collected data is not strong. The maximum accuracy is still under 65%. However, the models do show signs of promise; particularly, in that they universally outperform traditional machine learning methods, such as logistic regression. Of the network architectures explored, the 11 layer network without dropout performed the best. This indicates that the model inherently has more problems with bias than with variance.

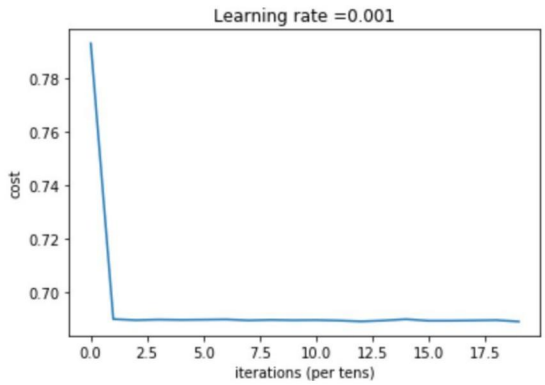

Fig. 8. The cost function of the three layer network with dropout over 100 epochs.

The reason for this poor performance is likely due to the dataset itself. The individual events (log data from the honeypot sensors) are noisy and may not contain enough information, especially with the encoding. While encoding allows us to avoid scarce networks, it requires a high level of expertise and will inevitably influence the model results.

The high bias observed in the model is likely also caused in part by the data itself. Collecting raw internet data will always introduce bias at some level. The collection of raw and biased internet data is therefore the cause of some of the performance degradation.

V. Future Work

When taking this project forward, there are a number of improvements we seek to make. The most important improvement to this project is more and better quality data. Currently, we believe the shorfalls of our exisitng model derive from a lack of quality data. Becasue we are using a self generated data set which contains only honeypot events with imperfect classification there are many improvements that can be made. Similarly, the balance between data encoding and scarce one hot vectors should be improved. Future iterations of models should look at various combinations of encoding methods. Moreover, there are other known indicators of potentially threatening behavior for IP addresses not listed on the blacklist. We did not consider these types of behaviors when labeling our data. This additional labeling might improve the results of our current model. It is possible that our current model predicts and detects threatening behavior from IP addresses listed as threats and the same behavior from unlisted IP addresses also as threats. However, in its current form, these are counted as false positives.

VI. Contributions

Isaac and Giovanni each contributed equally based on their respective skill sets. Isaac primarily worked with the data and developed the original network architecture, while Giovanni improved on this architecture in future iterations.

Both contributed equally to the poster creation and the writing of the paper.

VII. Code

The code and data for our project is publicly available on MatrixDS at https://community.platform.matrixds.com/project/5afb09f087c54574a2be3ce4. The project folders are organized as follows:

- DataCleaningAndPrep: All code for preparing the data for modeling
- DeepLearningFiles: All model notebooks and data
- ProjectReport: All report files

Please be patient for the project to render as there are quite a few files. Thanks!

References

[1] T Mahmood and Uzma Afzal. Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools. In Information assurance (ncia), 2013 2nd national conference on, pages 129134. IEEE, 2013.

[2] Javaid, Ahmad, et al. "A deep learning approach for network intrusion detection system." Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[3] Wang, Wei, et al. "Malware traffic classification using convolutional neural network for representation learning." Information Networking (ICOIN), 2017 International Conference on. IEEE, 2017.

[4] Ma, Tao, et al. "A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks." Sensors 16.10 (2016): 1701.

[5] Wang, Wei, et al. "End-to-end encrypted traffic classification with one-dimensional convolution neural networks." Intelligence and Security Informatics (ISI), 2017 IEEE International Conference on. IEEE, 2017.