Malware Detection with the EMBER Dataset using a Deep Learning Approach

Will Macdonald and Brennan Mosher

***Abstract*—Machine learning techniques, specifically deep learning, are being applied in a number of fields due to the recent increases in modern computational power. In this report, two methods for classifying the EMBER dataset are proposed; deep neural networks and LightGBM decision tree boosting. The neural network approach appears to be infeasible for the EMBER 2017 dataset, while LightGBM boosting will require future work to decide on the feasibility. Both these methods are presented, and distributed through Github, to allow anyone to continue the work. By implementing machine learning practices in malware detection, the hope is that this technique can give an improvement on signature-based malware detection, which require manual revision and cannot detect emerging cyber threats.**

1 Introduction

The computational power of everyday devices is growing exponentially, and with increase in capability comes more powerful algorithms. The development of machine learning (ML) and deep learning (DL) has seen significant improvements in the field of cybersecurity; and specifically significant improvement to malware detection [1]. There is, however, a duality to the benefits provided by these algorithms when considering cybersecurity. Not only does the introduction of DL improve the security implementations but also lends itself to allow adversaries to create more capable attacks [2].

In this report, two ML and DL approaches are taken with the goal of implementing more capable malware detection systems. The initial approach saw the implementation of a deep neural network for a binary classification task in which the goal was to distinguish correctly whether the file is benign or malicious, given features extracted from portable executables (PEs) files. This initial approach was then followed by the development of a Light Gradient Boosting Machine model for the same predictive task of binary classification for benign and malicious files. The reasoning behind the development of this second classification model was two fold; first, to address difficulties uncovered in the ability of the deep neural network to learn the underlying relations in the data; secondly, in order to compare how the two models evaluated on the dataset and determine which methodology is better suited to the prediction task at hand. An iterative approach to constructing the deep neural network was taken, such that first a minimum viable model was developed, followed closely by the evaluation and fine-tuning of the performance on subsets of the training dataset until a proper configuration was found.

Following the implementation of the deep neural network came the implementation of a LightGBM Boosting model was developed. This implementation saw a similar iterative approach to that of the deep neural network. First a baseline model was developed to be evaluated and fine-tuned on a training data subset to determine the optimal configuration before finally training on the full dataset.

The iterative approach to the development of these models was critical in finding the correct configuration for the model before training on the full dataset due to its sheer size and computational load that would be required from the development devices. For further information about the technical implementation of these algorithms discussed please visit the code repository as seen in https://github.com/willmacd/dl-malware-detection.

2 Dataset

The dataset selected for the research task of ML/DL malware detection was the Endgame Malware Benchmark for Research (EMBER) dataset; a labelled benchmark dataset for training machine learning models to statistically detect malicious portable executable (PE) files [3]. In aims to gain a more in depth understanding of the dataset,the academic paper "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models" [3] that was released in pair with the open source dataset proved quite insightful. This paper highlighted key information about the structure of the dataset and approaches taken by the authors of the dataset in order to develop a malware classification model of their own. This gave perspective into alternative approaches to the problem and understanding the contents and distribution of the data in EMBER.

EMBER is composed of pre-extracted features from 1.1 million portable executable (PE) files. The distribution of the dataset is as follows: 400,000 instances of malicious file features, 400,000 instances of benign file features, and 300,000 instances of unlabelled file features. The pre-extracted features are represented in an array of length 2381 in which each index represents its own feature from the file. These extracted features are explained more in depth in the EMBERs academic paper [3]. Further exploration of the dataset distribution revealed the training dataset is composed of 900,000 data instances, sampled evenly from benign, malicious and unlabeled classes (i.e., 300,000 instances of each). The testing dataset followed a similar even distribution however, it excluded instances from the unlabelled data class; testing dataset was composed of 200,000 instances sampled evenly (i.e., 100,000 instances from both benign and malicious classes). Further

visual representation of this data distribution can be seen in Figure 2.

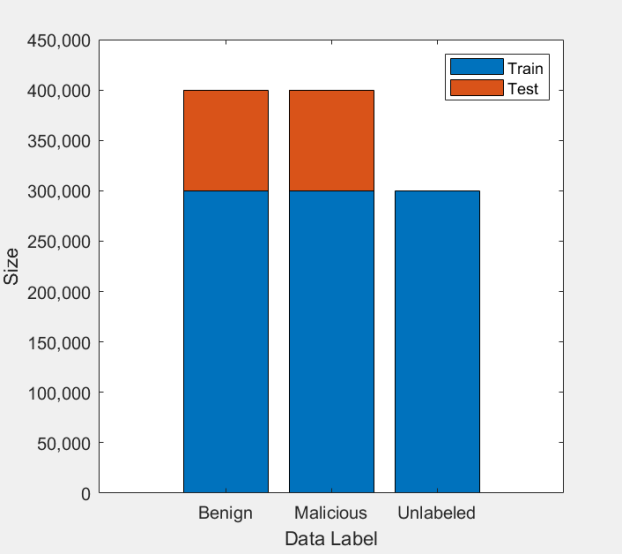

Fig. 1. Distribution of Dataset

Prior to the initial development of the ML/DL classification models, several data preprocessing steps were taken. To begin with the data preprocessing phase, the data was first converted from .json to a .dat file format such that built-in functionality from EMBERs open-source library [3] could be used to load the dataset into code. Once loaded into code, the data was then to be converted into tensorflow batch dataset format such that it could easily be interfaced with the deep neural network proposed. In the creation of these batch datasets, based on the format the data is presented, it was required that the data samples and their respective labels be separately converted to tensors and concatenated together during the development of the datasets.

Once the creation of these dataset was complete. A final operation in data preprocessing was required; the training dataset contained an equal part of unlabelled data, which effectively provides non-useful information to the supervised learning binary classification task at hand and therefore, all unlabelled data points from the dataset were dropped such that only instances that have explicit classes are considered.

3 Deep Learning Approach

i) Model

The initial approach proposed for the predictive task of classifying files as either malicious or benign was to implement a deep neural network for binary classification. More specifically, the minimum viable target network initially designed was a multi-layer forward-feeding network with an input layer accepting tensors with a feature space of 2381. Following the input, the data was fed into two hidden fully connected dense layers, the total number of neurons per hidden layer followed the general rule of thumb - hidden neurons should be $2/3$ the size of their input layer plus the size of the output layer. Finally, the hidden layers lead to an output layer with 2 neurons (one per class in the binary classification task). This approach was a naive implementation and, as will be explained in §5, resulted in poor performance.

This lacking performance was expected by the team, as this was simply the minimum viable model; therefore, model architecture was further iterated upon to make a deeper network. The goal of the deeper model was to learn more complex relations within the data at the cost of computational complexity. Not only was the second iteration deeper but also saw the implementation of specialized layers such as dropout layers to effectively combat overfitting in the model by randomly dropping neurons within the network (i.e., forcing the data to take different paths through the network at each epoch). The architecture of this second approach was once again a forward-feeding network with an input layer to accept tensors with the feature space of 2381. This architecture differed from the original, however, by implementing 4 hidden layers, following the same rule of thumb for number of neurons per layer as stated above. Each of which directly fed into a dropout layer before connecting onward to the next fully connected dense layer. Finally, the hidden layers led into an output layer with 2 neurons (corresponding to the binary classification classes).

The task at hand required for specific activation and loss functions to be considered throughout the development of this network. As stated, the predictive task is binary classification and therefore the loss function that the deep neural networks used in learning was binary cross-entropy, sigmoid activation functions were also implemented such that the outputs of the network are mapped to a range from 0 to 1.

i) Data Normalization

To overcome the issues of the network getting stuck at local optima in the initial model, data normalization techniques were used to improve the dataset for use in deep learning. Neural network models are often very sensitive to non-normalized data, so normalization techniques are commonly used to improve convergence.

The two most common data normalization techniques are standardization and normalization. Standardization scales data features so that the distribution is centered around zero with a standard deviation of 1. Normalization condenses the data into the range of 0 to 1, or in the case of the EMBER dataset -1 to 1 due to negative values in the features. The results of a sample of these techniques are shown in Table 1.

The issue with these normalization techniques for the EMBER dataset is the prevalence of large outliers. Within the dataset there exists outliers that are magnitudes larger than the other values in the dataset. These values are not distributed in a way that removing them would be possible without losing a large amount of information from the dataset. So these outliers heavily influence the normalization techniques and result in unusable data.

Robust scaling is a normalization technique that is designed with the purpose of working on data with outliers. Instead of

TABLE I
DATA NORMALIZATION OUTPUT

Normalization	Standardization	Robust Scaling
-0.021	1	308.514
-0.021	1	88.750
-0.021	0	82.461
-0.021	0	84.690
-0.021	1	84.236
-0.021	0	79.357
-0.021	-1	80.407
-0.021	-1	21021.504
-0.021	1	-21021.504

binding the data into the range -1 to 1, the scaler binds it within the interquartile range. This is the range between the first quartile and third quartile. This removes the effect of the outliers but results in outliers still remaining in the dataset. The sample output of this technique is shown in Table 1. As can be seen, the variance of data is retained at the cost of outliers being kept in the data.

iii) Principal Component Analysis

The original dataset contains 2381 features for each sample. Using principal component analysis (PCA) this can be reduced while keeping above 90% of the variance of the data. This technique would lower the computational load of training the neural network as well potentially mitigating the prevalence of outliers. For PCA to work, the data must be normalized/standardized, which is covered in §3.I. The PCA algorithm performs matrix operations and calculates eigenvectors and eigenvalues to determine the number of important features to retain [4].

To implement this technique the sklearn.Decomposition.IncrementalPCA Python package was used with the normalized data. To keep $>$90% of data variance the PCA computation resulted in 100 features of the original dataset being kept.

4 Gradient Boosting Approach

To overcome the challenges of using the dataset in a deep learning approach a second model was developed. This model was created using the LightGBM gradient boosting framework [5] for Python. Decision tree boosting was considered as the data would not require normalization due to the qualitative nature of decision tree algorithms [6]. The technique uses a forest of decision trees as an ensemble model to try to minimize total error. The first trees fit the data with a simple model and analyze the error. Next, more trees are fit to the data with the goal to minimize the error of the past trees. Specifically LightGBM uses a gradient descent boosting algorithm which works on the loss function of the algorithm, instead of the error function. The problem was again formulated as a binary classification problem, which decision tree models are well suited to perform [7]. The same 600,000 sample (evenly distributed, excluding unlabelled instances) dataset was used, as in the deep learning model, to complete this task. Due to the nature of decision trees, the data was not normalized.

TABLE II
PARAMETERS USED IN CONSTRUCTION OF LIGHTGBM GRADIENT BOOSTING MODEL

Parameter	Value
Number of Leaves	2048
Max Tree Depth	15
Minimum Data in Leaf Node	50
Learning Rate	0.05

The LightGBM package has built in training functionality [8], so this was used as the base and the parameters of the model were tuned according to the error function. The final architecture of the boosting model is shown in Table 2.

To evaluate the model the lightGBM.Booster.predict method was used with a threshold of 0.5. This threshold decides on the output classification from the prediction probability returned by this method. The threshold was chosen to be 0.5 since the dataset is balanced, so there is no need to raise/lower the threshold to account for unbalanced data.

5 Evaluation

The two methods introduced in this paper - deep neural network and light gradient boosting machine - were evaluated across two devices: a Microsoft Surface Pro 4 with an Intel Core i7-6650U CPU and 8 GB of RAM and Lenovo T470 with an Intel Core i5-7200U and 8GB RAM. While evaluating the deep neural network approach, the model faced significant difficulty throughout training across a large range of tested hyperparameters, including but not limited to changes in learning rate, structure (i.e. number of layers and neurons per layer), implementation of dropout, changes to activation functions and optimizers implemented, and so forth. The difficulty that the deep neural network faced, took the form of training and validation accuracy converging to 50.5% and having the loss function remain at local optima of 0.632 across a various instantiations with random initial weights in the network; effectively inhibiting the model from further learning relations within the data.

One large contributing factor that could have caused such behaviour was the fact that neural networks have a significantly more difficulty learning from unnormalized data and as shown in the §3.I in Table 1. Due to the significant variance across all data values and the skewness of the data this was a large contributing factor to the poor performance achieved in this approach.

Using the LightGBM boosting model the training accuracy reached 99.8% but had difficulties generalizing, achieving only 50.5% testing accuracy. This test accuracy is not a significant improvement on the binary classification problem with random guesses. This behaviour in a predictive model is representative of the model overfitting to the data; in turn minimizing its ability to generalize outside of the data the model was trained on.

Due to time constraints on the project the model was not able to be optimized to the dataset. Tuning the hyperparameters of the model is a common technique used to reduce overfitting in decision tree boosting models. Many of the parameters were changed to reduce overfitting, these included depth of trees, data in leaves, and number of leaves in a tree. These techniques did not improve the ability for the model to generalize on new data. So future work will be to improve the performance of the model to a useful accuracy, ideally greater than that of random guessing.

6 Conclusion

As shown by our experiments a deep learning approach to classifying the EMBER dataset does not seem feasible due to the nature of the data. The outliers and inability to normalize the data limits the application of neural networks on the large dataset. Though the LightGBM model proposed was not successful in classifying the data, it is believed that moving forward with this approach may be a feasible approach to the predictive task at hand.

7 Future Work

EMBER has since released two new datasets for classifying malicious files: EMBER 2017 v2 and EMBER 2018. Using a machine learning approach similar to those presented in this report is an interesting avenue that may have success. Different criteria of data features are included in the other datasets [3] which may be more compatible with the models.

Continuing work on the LightGBM boosting model is also an interesting direction to continue this project. The constrained timeline of implementation, due to the work on implementing the neural network, limited the possibility of exploring this deeply. As well, continuing with ensemble learning [9] by implementing a model through a similar gradient descent boosting algorithm XGBoost is interesting. For future work related to applications of the proposed model would be developing a working real-time detector. This would scan files/executables on the users computer then predict using the model to output the classification to the user. This implementation would be the next stage in development if this project timeline had allowed.

8 References

[1] I. Firdausi, C. Lim, A. Erwin, and A. Satriyo Nugroho, "Analysis of Machine Learning Techniques Used in Behavior-Based Malware Detection," IEEE, 2010.

[2] R. Kulshrestha, "Malware Detection Using Deep Learning," 01-Jul-2019. [Online]. Available: https://towardsdatascience.com/malware-detection-using-deep-learning-6c95dd235432. [Accessed: 11-Apr-2021].

[3] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," ArXiv e-prints2, 2018.

[4] H. Abdi and L. J. Williams, "Principal Component Analysis," Wiley Interdisciplinary Review: Computational Statistics, vol. 2, 2010.

[5] Microsoft, "LightGBM Documentation," LightGBM 3.2.0.99 documentation, 2021. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/. [Accessed: 11-Apr-2021].

[6] J. Gareth, D. Witten, T. Hastie, and R. Tibshirani, "An Introduction to Statistical Learning with Applications in R," Springer Texts in Statistics, vol. 103, 2013.

[7] Y.-yan Song and Y. Lu, "Decision tree methods: applications for classification and prediction," Shanghai Archives of Psychiatry, vol. 27, no. 2, 2015.

[8] Microsoft, "Lightgbm.train," LightGBM 3.2.0.99 documentation, 2021. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.train.html. [Accessed: 11-Apr-2021].

[9] XGBoost Developers, "XGBoost Documentation," XGBoost Documentation - xgboost 1.4.0-SNAPSHOT documentation, 2020. [Online]. Available: https://xgboost.readthedocs.io/en/latest/. [Accessed: 11-Apr-2021].