ICPR 2020
25th INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION
Milan, Italy 10 | 15 January 2021

NUS National University of Singapore

Malware Detection by Exploiting Deep Learning over Binary Programs

Panpan Qi*, Zhaoqi Zhang*, Wei Wang* and Chang Yao†

*School of Computing, National University of Singapore, Singapore
†Institute of Computing Innovation, Zhejiang University, China

Outline

1. Motivation
2. Related Work
3. Contribution
4. Preliminary
5. Methodology
6. Experiments
7. Conclusion

Motivation

- Malware (malicious software) remains the most popular and damaging attack vector, costing hundreds of billions in damage [1].
- Malware evolves rapidly, with reports showing that 99% disappear after 58 seconds [2].
- Traditional machine learning models heavily depend on feature engineering and could be easily deceived by hackers.
- In practical applications, the anti-virus industry prefer to increase the recall (i.e. true positive rate) while maintaining a low false positive rate (usually less than 0.1%).
    - Recall: ratio of the malware correctly identified as malware.
    - False positive rate: ratio of benign software incorrectly identified as malware.

Related Work

Commercial Antivirus Solutions

- McAfee, Symantec, TrendMicro, and online services like VirusTotal etc.
- Low accuracy and high memory overhead
  - Too much dependence on **large** databases of malware **signatures** (a regular expression string that can be matched by scanning the malware).
  - Malware signatures are easily bypassed by encrypting the payload, or by other obfuscation techniques like polymorphism and metamorphisms.

Related Work

Research work

- **Feature engineering**

  Static analysis: features are directly obtained without running it (e.g. opcodes, printable strings, n-grams, system API calls, control flow graph, import tables etc.).

  - **Advantage**
    - It does not require complex or expensive setup for collecting, and is able to avoid the complications caused by running samples.
  - **Limitation**
    - Static analysis are more vulnerable to code obfuscation.

Related Work

Research work

- **Feature engineering**

  Dynamic analysis: features are obtained by running the sample in an isolated environment and monitoring its run-time behavior. (e.g. instructions and system call sequences etc.)

  - **Advantage**
    - Intuitively, malicious behavior is the best indicator of a malicious sample.
  - **Limitations**
    - It requires much computational resource as the analysis must run long enough to capture meaningful behavior;
    - Certain malwares can detect the dynamic execution and prevent itself from exhibiting any malicious behavior.

Related Work

Research work

- **Classifiers**

Traditional Machine learning models

- Decision Trees (DT)
- Naive Bayes (NB)
- Boosted DT
- Boosted NB
- Support vector machines (SVMs)
- ……

Deep learning models

- MLP(Multilayer Perceptron)
- CNN(Convolutional Neural Network)
- Autoencoder based Neural Network
- LSTM
- ……

Related Work

- Existing research works using machine learning claim very high detection rates of over 90%, far better than incumbent antivirus products.
- Failed to gain traction in the industry because
  - Malware **evolves rapidly** as malware creators find new ways to exploit or to evade existing defense.
  - Although most models achieve less than **1% false positive rates**, these rates are still considered too high to be useful in practice.

Contribution

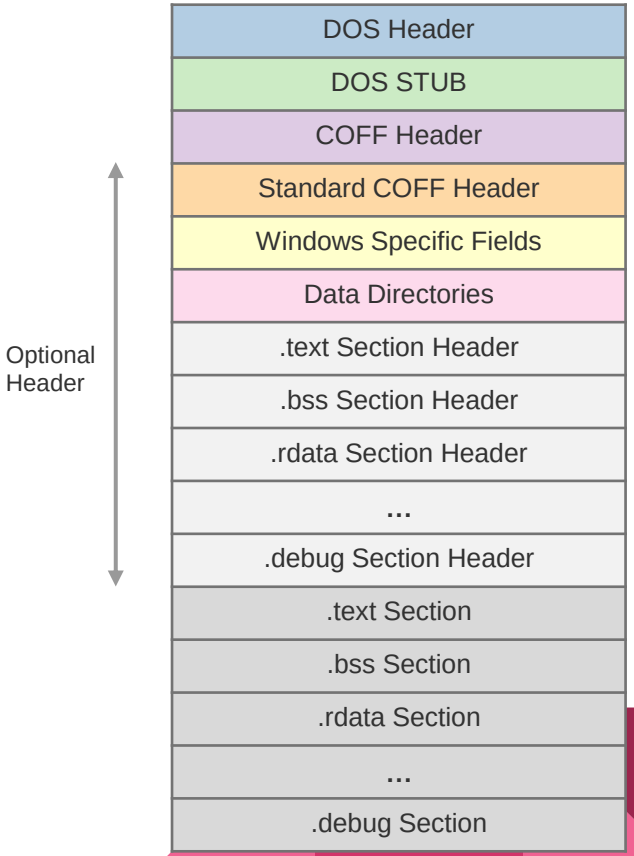
- Proposed an **end-to-end malware detection framework** based on deep learning techniques, which achieves the best performance among existing deep learning based methods.
- Proposed an effective **loss function** for optimizing recall with a fixed tiny false positive rate.
- Conducted experiments on **a real large dataset** to confirm the effectiveness of the proposed feature learning framework and loss function for malware detection.

Preliminary: PE format

The **Portable Executable (PE) format** is a file format for executables, object code, DLLs, FON Font files, and others used in Windows operating systems.

PE Header

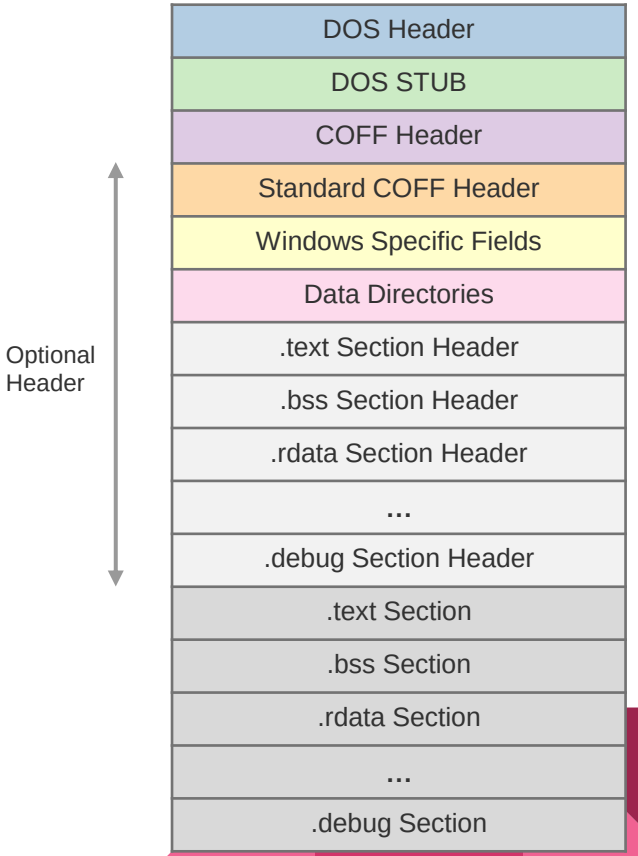
- Consists of DOS header, DOS STUB, COFF Header and optional Header
- Contains the most basic and meaningful information about the file
  - the target machine types
  - the number of the sections
  - initial stack size
  - preferred base address
  - operating system version
  - …



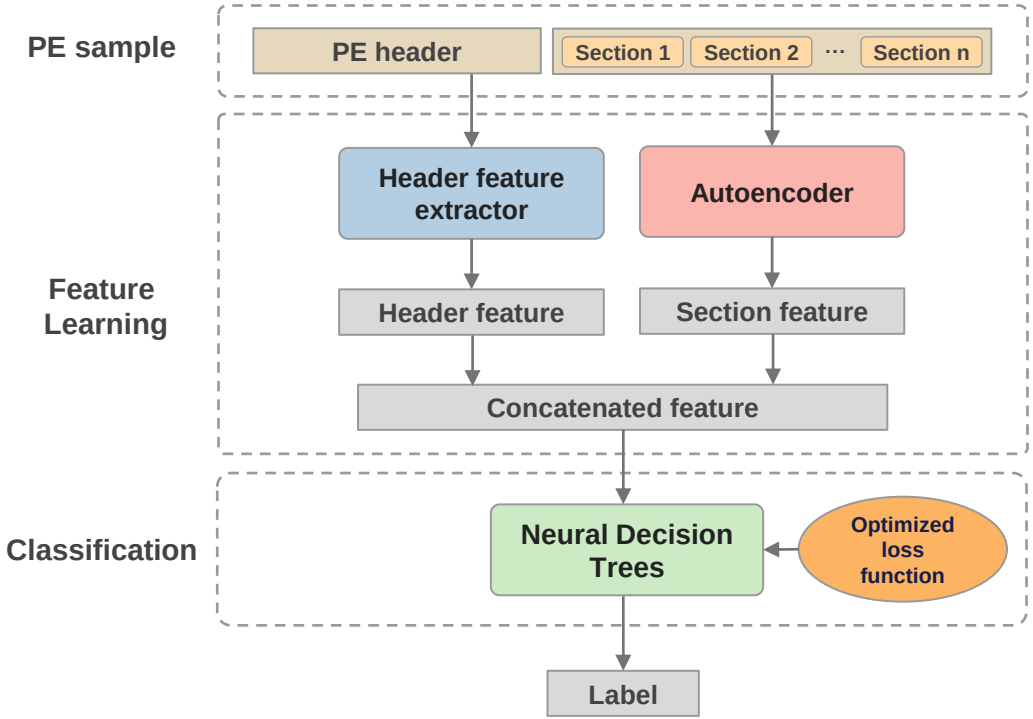
Preliminary: PE format

Section

- Contains the main content of the file, including code, data, resources and other executable files.
- Vary in length (usually very long) and the information is **scattered** throughout.
- Previous work does not pay enough attention to the section part.
  - N-gram
  - Byte entropy histogram
  - String information
  - …



Methodology



An end-to-end model for malware detection

Header Feature Extraction

Input

- Raw byte sequence of the PE header

Embedding layer

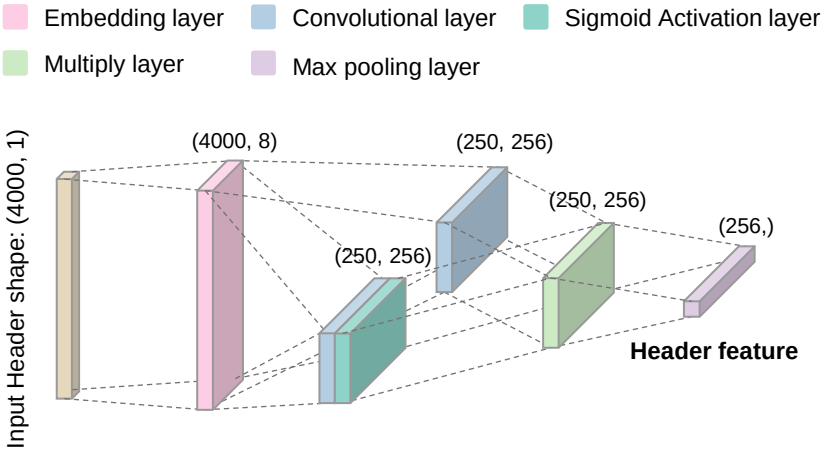
- Embeds the raw bytes into a continuous and distributed representation

Gated Convolution layer

- $X_A \otimes \sigma(X_B)$
- Provides a mechanism to learn, select and pass along the important and relevant information.

Global Max-pooling layer

- Produces the activation(the header feature) regardless of the location of the detected features.



Section Compression

Input

- Multiple executable sections in a PE sample

Encoding:

- Convolutional layer + 1D Max pooling layer

Decoding:

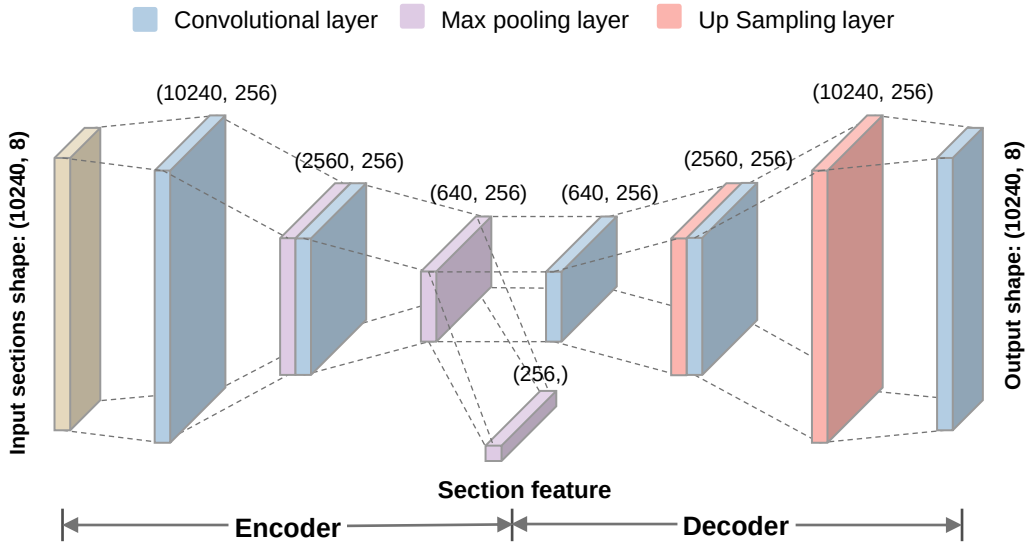
- Convolutional layer + 1D Up sampling layer

Loss function

- MSE(Mean squared error):

$$L(X_S, Z) = \frac{1}{n}\sum_{i=1}^{n}\left(X_{S_i} - Z_i\right)^2$$

where X_S is the input sections and Z is the observed output.



Neural Decision Trees

A differentiable version of decision tree

- Designed by Kontschieder et al.[3]
- Follows the classical full binary tree structure.

For each decision node $d \in \mathcal{D}$

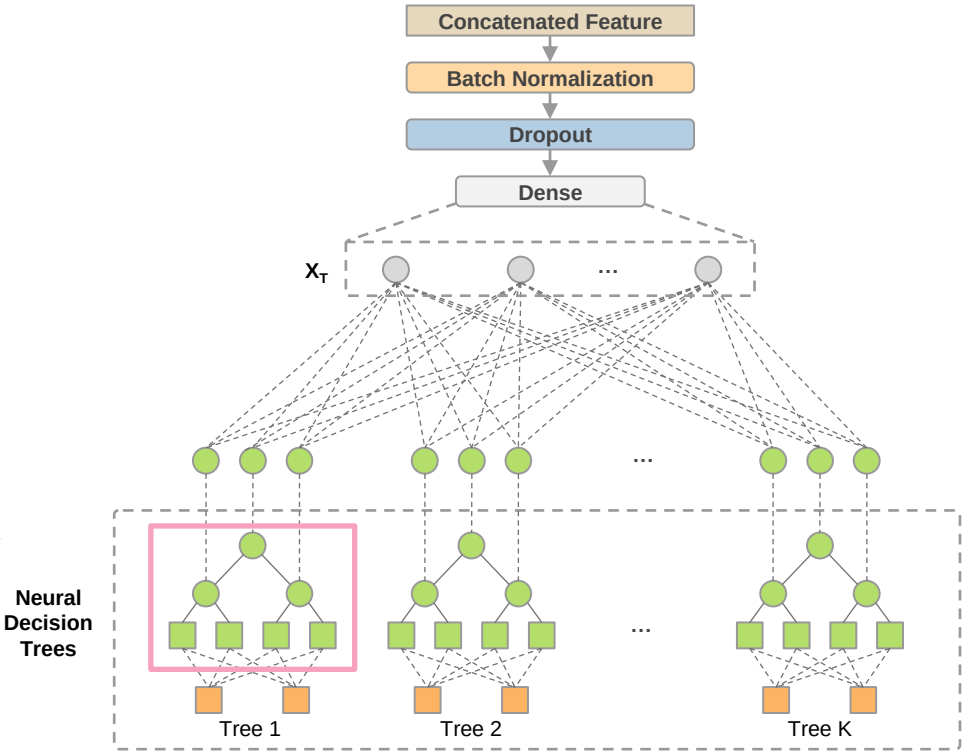
- Holds a decision function, the probability that a sample reaches node d and be sent to the left subtree.

$$D_d(X_T) = \sigma\big(f_d(X_T)\big) \in [0, 1]$$

where f_d is the transfer function, $f_d(X_T) = W_T X_T + b_T$.

For each leaf node $l \in \mathcal{L}$

- Holds a probability distribution P_l over the labels.
- P_{l_Y} stands for the probability for the samples in leaf l predicted to be label Y.



Neural Decision Trees

- The probability of a sample predicted as label Y by tree k is

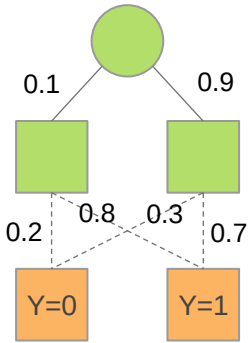
$$\mathbb{P}_{T_k}[Y|X_T] = \sum_{l \in \mathcal{L}} P_{l_Y} \prod_{d \in \mathcal{D}} \left(D_d(X_T)^{\mathbb{I}_{left}} \overline{D_d}(X_T)^{\mathbb{I}_{right}} \right)$$

where $\overline{D_d}(X_T) = 1 - D_d(X_T)$, $\mathbb{I}_{left}$ is the indicator function for the sample that will be sent to the left subtree.

e.g.

$$\mathbb{P}_{T_k}[Y = 0|X_T] = 0.2 \times 0.1 + 0.3 \times 0.9 = 0.29$$

$$\mathbb{P}_{T_k}[Y = 1|X_T] = 0.8 \times 0.1 + 0.7 \times 0.9 = 0.71$$



Tree k with depth 1

Neural Decision Trees

Bagging (Neural Random Forest)

- The prediction is made by averaging the outputs of all the trees

$$\mathbb{P}[Y|X_T] = \frac{1}{K}\sum_{k=1}^{K}\mathbb{P}_{T_k}[Y|X_T]$$

- Loss function: binary cross entropy

$$L_{NDT}(X_T, y) = -(ylog(\mathbb{P}[Y=1|X_T] + (1-y)\log(\mathbb{P}[Y=0|X_T]))$$

Neural Decision Trees

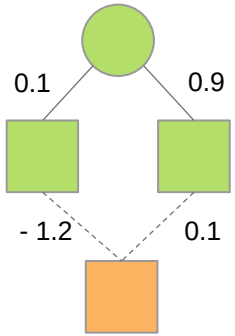
Boosting (Neural Gradient Boosting Decision Trees)

- The value of a sample predicted by tree k (a regression tree) is

$$\mathbb{P}_{T_k}(X_T) = \sum_{l \in \mathcal{L}} w_l \prod_{d \in \mathcal{D}} \left(D_d(X_T)^{\mathbb{I}_{left}} \overline{D_d}(X_T)^{\mathbb{I}_{right}} \right)$$

e.g.

$$\mathbb{P}_{T_k}(X_T) = -1.2 \times 0.1 + 0.1 \times 0.9 = -0.03$$



Tree k with depth 1

Neural Decision Trees

Boosting (Neural Gradient Boosting Decision Trees)

Input : The input feature X_T, the true label y, learning rate ρ, the initial predicted value $f_0(X_T)$

Output : The predicted value $\mathbb{P}(X_T)$

$\ell_{NDT} \leftarrow 0.0$

for $k = 1$ **to** K **do**

  $r_k \leftarrow y - f_{k-1}(X_T)$

  $f_k(X_T) \leftarrow f_{k-1}(X_T) + \rho \cdot \mathbb{P}_{T_k}(X_T)$

  $\ell_{NDT} \leftarrow \ell_{NDT} + \rho \cdot mse(r_k, \mathbb{P}_{T_k}(X_T))$

end

$\mathbb{P}(X_T) = f_K(X_T)$

Neural Gradient Boosting Decision Tree Algorithm

Neural Decision Trees

Logistic regression

- Apply logistic regression on all the outputs of the decision trees
- Enable a more flexible way of utilizing the generated trees
- The final output is

$$\sigma(W_{LR}X_{LR} + b_{LR})$$

where X_{LR} denotes the outputs of all the single trees and $\sigma(x)$ is the sigmoid function.

- The loss function is also taken as the binary cross entropy

Loss function of the model:

$$L = L_{AE}(X_S, Z) + L_{NDT}(X_T, y) + L_{LR}(X_{LR}, y)$$

Loss Function Optimization

- To maximize recall with the restriction that false positive rate $\leq 0.1\%$.
  - Recall: ratio of the malware correctly identified as malware.

$$TPR = \frac{tp}{tp + fn} = \frac{tp}{|Y^+|}$$

  - False positive rate: ratio of benign software incorrectly identified as malware.

$$FPR = \frac{fp}{fp + tn} = \frac{fp}{|Y^-|}$$

- The maximum TPR with at most α FPR problem can be defined as

$$\max_f \frac{tp}{|Y^+|} \; s.t. \frac{fp}{|Y^-|} \leq \alpha$$

- We can rewrite tp and fp by the zero-one loss:

$$\max_f 1 - \frac{\sum_{i \in Y^+} l_{01}(x_i, y_i)}{|Y^+|} \; s.t. \frac{\sum_{i \in Y^-} l_{01}(x_i, y_i)}{|Y^-|} \leq \alpha$$

True / Predicted	Positive	Negative
Positive	tp	fp
Negative	fn	tn

Confusion matrix

Loss Function Optimization

- Since zero-one loss is **non-convex** and **not smooth**, we lower bound tp and upper bound fp by its approximate upper bound, the log loss. :

$$\min_f \frac{\sum_{i\in Y^+} l(x_i, y_i)}{|Y^+|} \; s.t. \frac{\sum_{i\in Y^-} l\,(x_i, y_i)}{|Y^-|} \leq \alpha$$

- Applying Lagrange multiplier theory, the optimized loss function is

$$L = \frac{\sum_{i\in Y^+} l(x_i, y_i)}{|Y^+|} + max\left(0, \lambda\left(\frac{\sum_{i\in Y^-} l\,(x_i, y_i)}{|Y^-|} - \alpha\right)\right)$$

Loss for positive samples

Loss for negative samples

Experiments

Data Summary

- Provided by **SecureAge**, with granularity at the monthly level.
- SecureAge deployed 12 commercial antivirus engines that are continuously scanning data from the endpoints.
  - Positive: num of engines >= 4
  - Negative: num of engines = 0

Dataset	Positive samples	Negative samples
February	110656	80185
March	100651	92097
April	58394	48595
May	42635	87858

Summary of the data

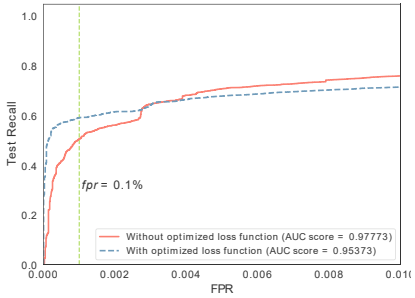
Experimental Results

- The proposed model achieved the best AUC score, and recall when fpr <= 0.1% among all the models without hand-crafted features
- Models with the derived optimized loss function generally outperform those without the optimized loss function.

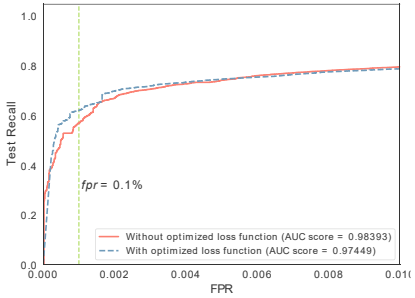
Training Dataset	Test Dataset	Model	Without optimized loss function		With optimized loss function	
			AUC (%)	Recall (%)	AUC (%)	Recall (%)
February	March	MalConv [4]	95.45±0.34	33.58±16.21	94.79±0.32	53.17±4.37
		ConvNet [5]	96.21±0.17	45.11±3.88	94.34±0.60	49.92±3.69
		EntropyNet [6]	91.61±0.22	33.88±9.13	88.13±0.73	41.52±4.38
		Proposed Model	**96.47±0.20**	**56.14±3.65**	**96.40±0.19**	**57.52±2.95**
March	April	MalConv	98.50±0.12	50.67±11.75	98.21±0.31	57.41±9.74
		ConvNet	98.82±0.12	63.67±5.50	98.27±0.70	67.39±5.69
		EntropyNet	95.70±0.32	24.53±6.76	93.95±0.48	49.68±8.09
		Proposed Model	**99.16±0.04**	**71.54±3.32**	**99.12±0.07**	**75.25±1.62**
April	May	MalConv	97.95±0.36	52.28±8.12	94.02±1.48	58.55±2.43
		ConvNet	98.33±0.26	55.91±2.68	96.66±0.73	56.96±3.45
		EntropyNet	90.96±0.96	31.33±3.13	81.94±2.33	35.23±3.24
		Proposed Model	**98.60±0.20**	**70.29±1.03**	**98.43±0.35**	**70.69±0.93**

Experimental Results

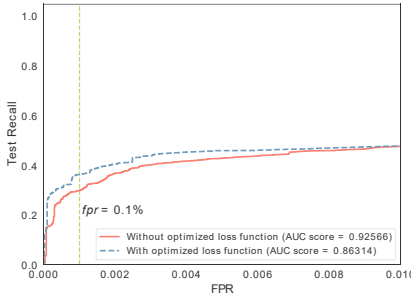
- The proposed model achieved the best AUC score, and recall when fpr <= 0.1% among all the models without hand-crafted features
- Models with the derived optimized loss function generally outperform those without the optimized loss function.



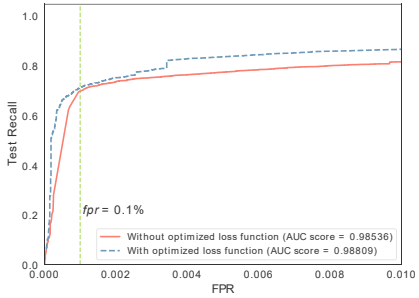
(a) ROC curves of MalConv



(b) ROC curves of ConvNet



(c) ROC curves of EntropyNet



(d) ROC curves of Proposed Model

Experimental Results

Ablation study

- The addition of each component brings an improvement to the performance.

Autoencoder	Neural Decision Trees	Logistic Regression	AUC (%)	Recall (%)
No	No	Yes	98.89±0.12	68.99±1.51
Yes	No	Yes	98.86±0.05	69.33±1.75
Yes	Neural Random Forest	No	98.70±0.19	69.89±1.86
Yes	Neural GBDT	No	98.68±0.26	69.75±2.44
Yes	Neural Random Forest	Yes	98.92±0.11	70.10±1.53
Yes	Neural GBDT	Yes	98.60±0.20	70.29±1.03

Conclusion

- We propose a hybrid end-to-end framework for malware detection with an autoencoder and the Neural Decision Trees.
- We derive an optimized loss function to improve recall when fp rate $\leq 0.1\%$;
- The framework can be regarded as a further exploration to minimize the use of the domain knowledge in malware detection task.
- Experimental results demonstrate that the proposed framework is effective for malware detection.

References

[1] John F Gantz, Richard Lee, and Alejandro Florean. The Link between Pirated Software and Cybersecurity Breaches. National University of Singapore & IDC, 2014.

[2] Verizon. 2016 Data Breach Investigations Report. Technical Report 1, 2016.

[3] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo.Deep neural decision forests. InProceedings of the IEEE international conferenceon computer vision, pages 1467–1475, 2015.

[4] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. Malware detection by eating a whole exe. arXiv preprint arXiv:1710.09435, 2017.

[5] Marek Krčál, Ondřej Švec, Martin Bálek, and Otakar Jašek. Deep convolutional malware classifiers can learn from raw executables and labels only. 2018.

[6] Gibert, D., Mateu, C., Planes, J., Vicens, R.: Classification of malware by using structural entropy on convolutional neural networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

THANK YOU