Autoencoder Based IDS

Krishna Vasudev, Huichen Li, Professor Bo Li
Department of Physics, College of Liberal Arts and Sciences, University of Illinois at Urbana-Champaign

INTRODUCTION

Background Information:

1. Organizations face risk of damaging malware through transmission of data packets, which can alter/steal info.
2. Study Focus: Using autoencoders to detect malicious attacks
3. Two Phase Neural Network connected via transfer learning
   - Autoencoder: Compress high dimension input
   - Feed Forward Neural Network (FNN): Generate Class probabilities
4. Autoencoder Efficiency: How efficiently an autoencoder can be trained. Depends on model specifications.

KDD-NSL Dataset Attack Types:

1. Normal: Normal Operation
2. R2L: Onsite machine intrusion
3. DOS: Traffic flood to initiate shutdown
4. Probe: Gain network/organization information
5. U2R: Normal Operator gaining root access (i.e. admin)

Data Transformations:

Label Encoder			One Hot Encoder			
Name	Cat. #	Calories	Apple	Chicken	Brocolli	Calories
Apple	1	95	1	0	0	95
Chicken	2	231	0	1	0	231
Brocoli	3	50	0	0	1	50

Table 1: Distribution Merging Preprocessor

1. One Hot: Is this point an apple? If yes: Apple = 1
2. Label: Assign a label some number

THEORY

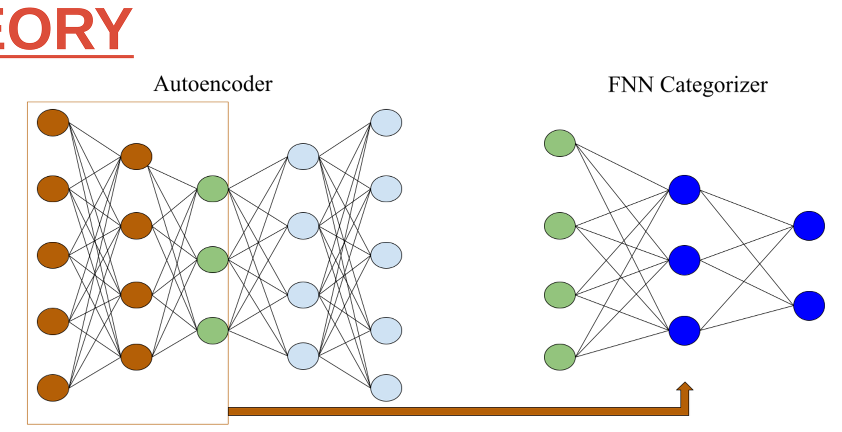

Figure 1: Autoencoder Based IDS Model

The model is made of 3 components:

1. Encoder that shrinks vector size
2. Decoder expands the vector size. Mean Square Error (MSE) between original vector and decoded vector is:

$$MSE = \frac{1}{N}\sum_i y_i - \hat{y}_i$$

3. Discard the decoder and attach an FNN categorizer. Train the FNN categorizer:
   - Generate encoded vectors using trained autoencoder
   - Use encoded vector as training input for FNN network.
   - Utilize Cross Entropy Loss (CEL) to generate class probabilities:

$$CEL = \frac{1}{N}\sum_i t_i \log(\, p_i)$$

EXPERIMENTAL METHODS

Pipeline #1:

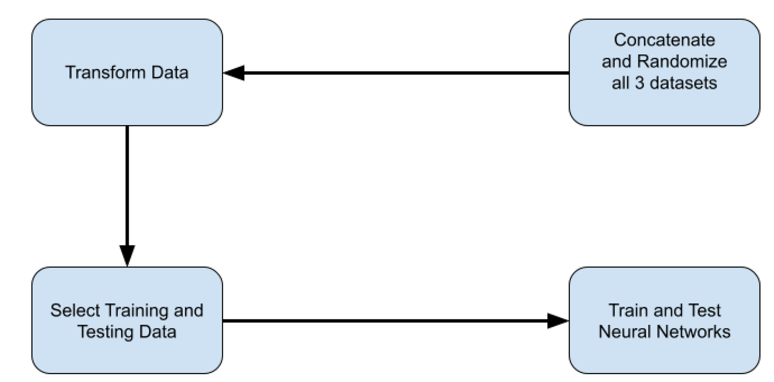

Figure 2: Distribution Merging Preprocessor

1. Concatenate and Randomize data (which merges distribution of datasets).
2. Transform datasets w/ OneHot, MaxAbsScaler (Normalize from 0 to 1) and Label Encoder
3. Select Training and Testing Data
   - Autoencoder Training Set: 100,000 random points
   - FNN Training Set: Max 100 points per class
   - Testing Set: 30,000 random points

Pipeline #2:

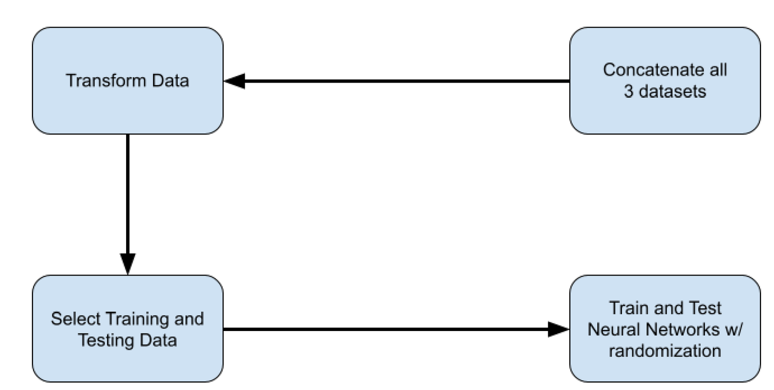

Figure 3: Non-Distribution Merging, Non-Sorter Preprocessor

1. Concatenate but do not randomize data (keeps distribution of datasets separated)
2. Next two steps are the same as Experiment 1. Randomize sets individually while training

Pipeline #3:

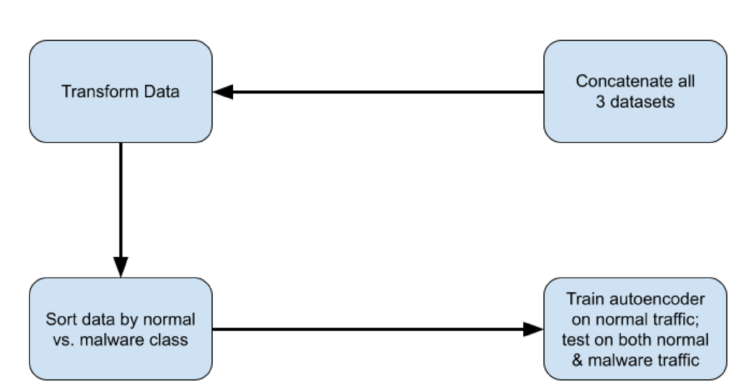

Figure 4: Non-Distribution Merging, Sorter Preprocessor

1. Concatenate but do not randomize data (keeps distribution of datasets separated)
2. Transform datasets w/ OneHot, MaxAbsScaler and Label Encoder
3. Sort data on normal or malware labels
4. Train autoencoder on normal traffic; Test autoencoder on both classes of traffic

RESULTS

Pipelines #1 and #2:

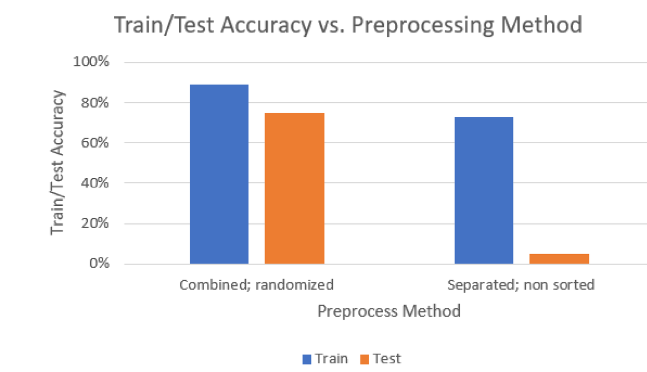

Figure 5: Train/Test Accuracy vs. Preprocessing Method

1. Back attack misclassification as normal traffic (Pipeline 1):
   - ~ 50,000 bytes transferred from source to destination
   - ~ 8,000 bytes transferred from destination to source
   - TCP protocol; Http Service; SF or RSTR flag

Pipeline #3: Separate Datasets; Sorted Data

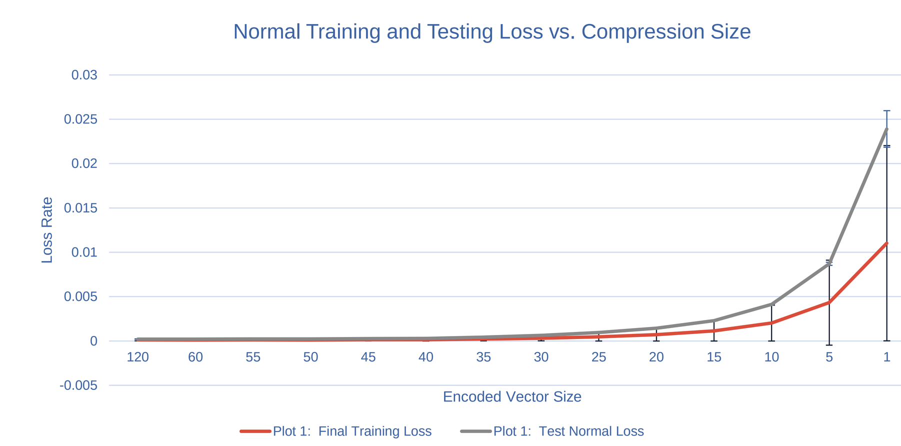

Figure 6: Normal Training and Test Loss vs. Compression Size

1. Exponential growth of losses as compression size of autoencoder decreases
2. High standard deviation and mean of testing losses

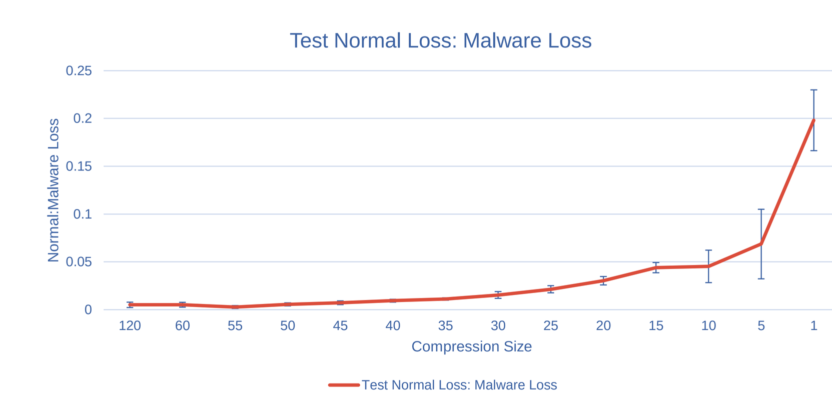

Figure 7: Normal Testing Loss of Malware

1. Similar to Figure 2, exponential loss is present and has a higher standard deviation as compression size decreases
2. Malware Losses throughout all compression sizes are higher by factor ~10

Figure 8: Normal Testing Loss of Malware

1. Note the dip in efficiency in parameter sizes. (Gap from 120 to 60)
2. Dip in efficiency potentially due to combination increased parameter count and parameters values other than 1.

CONCLUSIONS

Main Takeaways:

1. Autoencoder IDS learned the distribution of attack classes instead of the characteristics of the class.
   - Evidenced in Figure 5 when testing accuracy of non-distribution merging, non-sorter preprocessor decreased
   - Pipelines #1 and #2 are ineffective in identifying types of network traffic
2. From Figure 8, we observe a dip in training efficiency possibly due to:
   - High number of parameters to train
   - Multiplicative operations with matrices other than identity matrix.
3. From Figure 6 and 7, the autoencoder can distinguish between general malware and normal traffic
   - The normal and malware test loss differ by several standard deviations of one another

Further Actions:

1. Based off Pipeline #3, train a FNN that decides class labels in the following methods:
   - Progressive: Predict attack type mentioned in the introduction. Then predict specific class
   - Direct: Predict specific class w/o attack type

APPENDIX

Citations:

1. Javaid, Ahmad & Niyaz, Quamar & Sun, Weiqing & Alam, Mansoor. (2015). A Deep Learning Approach for Network Intrusion Detection System. EAI Endorsed Transactions on Security and Safety. 3. 10.4108/eai.3-12-2015.2262516.
2. Miller, D. J., Hu, X., Qiu, Z., & Kesidis, G. (2017). Adversarial learning: A critical review and active learning study. 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP). https://doi.org/10.1109/mlsp.2017.8168163
3. Solanki, S., Gupta, C., & Rai, K. (2020). A survey on machine learning based Intrusion Detection System on NSL-KDD dataset. International Journal of Computer Applications, 176(30), 36–39. https://doi.org/10.5120/ijca2020920343

ACKNOWLEDGEMENTS

I would like to thank the ALERT program and the Office of Naval Research for sponsoring this project.

ILLINOIS