Online Malware Detection with Variational Autoencoders

Jiří Tumpach[1,2], Martin Holeňa[2]

[1] Faculty of Mathematics and Physics, Charles University, Malostranské nám. 2, Prague, Czech Republic
[2] Institute of Computer Science, Czech Academy of Sciences, Pod vodárenskou věží 2, Prague, Czech Republic

Abstract: This paper studies the application of variational autoencoders (VAEs) to online learning from malware detection data. To this end, it employs a large real-world dataset of anonymized high-dimensional data collected during 375 consecutive weeks. Several VAEs were trained on selected subsets of this time series and subsequently tested on different subsets. For the assessment of their performance, the accuracy metric is complemented with the Wasserstein distance. In addition, the influence of different kinds of data normalization on the VAE perfomance has been investigated. Finally, the combinations of a VAE with two multi-layer perceptorns (MLPs) have been investigated, which has lead to the surprising result that the impact of such a combination on malware detection is positive for a simple and superficially optimized MLP, but negative for a complex and well optimized one.

1 Introduction

Neural networks are popular due to their performance, versatility, scalability and learning of features. Due to feature learning, a neural network, especially a deep one generally does not need extensive feature engineering. On the other hand, it requires a large amount of training data. There exist application areas where feature distribution and optimal classification can change in time. Classification in such situations is often called online classification. One of such areas is malware detection. The reason for changing distribution in malware detection is data drift, which comes from several sources. Firstly, benign programs are using different frameworks or statically linked libraries based on changing popularity. Secondly, malicious programs attempt to hide by copying clean program's code and behaviour, or using evading techniques like the manipulation of code during runtime, and those techniques are also changing in time, as a reaction on updating the databases of malware detection software. For this reason, neural networks hardly ever have enough training data to be promptly prepared for new threats. Moreover, even if they had enough data, their training would probably be too slow. This paper reports a work in progress investigating one of the possibilities to adapt neural networks for malware detection. We prepared a hybrid model that consists of multiple pairs of a generator and a classifier. Our idea is to make the generators learn the history and later use them for data augmentation because many malware evasion techniques are shared. As the generators, we use variational autoencoders (VAEs), and classification is performed using multi-layer neural networks. Firstly, we present results of our hybrid classification-autoencoder model, which employs a kind of transient feature learning. Secondly, we investigate how an autoencoder behaves when it is trained with high amounts of outliers.

In Section 2, we survey techniques for online malware detection. Section 3 is devoted to the principles of methods employed in our investigation. Section 4 present the results of performed experiments.

2 Online Malware Detection

Malware is continuously evolving by exploiting new vulnerabilities and examining evading techniques [7]. This causes significant changes in malware behaviour and consequently complicates the application of classification techniques for malware detection. These techniques must be precise enough not to bother user with false positives, as well as highly adaptive in order to detect new threats as soon as possible.

Malware detection techniques are static and dynamic [8]. While static methods are focused on an analysis of program code without actually running a particular program, dynamic methods analyse programs behaviour. Their main benefit is a potentially finite number of inspected behaviours. Commonly used features for dynamic analysis are resources, privileges, calls (system or APIs) and results of tracking sensitive data (e-mails, etc.) inside of an application [7].

One of the successful examples of the dynamic approach to malware detection is DroidOL [7]. It analyses inter-procedural control-flow sub-graphs with an adaption of the Weisfeiler-Lehman Kernel (WL). WL was successfully used to classify graph type with remarkable speed thanks to its linear time complexity. The authors of DroidOL found out that this representation is robust against hiding attempts of malicious software. After WL prepossessing, DroidOL applies

a passive-aggressive classifier, which is a kind of online learning methods suited for big data. On real Android applications, DroidOL outperforms state-of-the-art malware detectors.

Online Support Vector Machines with RBF kernel were investigated in [9], and they use features based on application behaviour, too.

A different approach to malicious software was investigated in [8]. The authors understand that allowing every application access to every information (location, contacts, pictures, ...) is not ideal. Alternatively, constantly bothering users by attributions of privileges could make them indifferent. Their system XDroid [8] tackles this problem by online hidden Markov model (HMM) learning users priorities.

This paper reliews on the content of [10] where the main idea is firstly introduced, now we try to identify the problems we had and try to come up with a solution that could be useful in similar situations. We are especially interested in the difficulty of VAE training since it is a model which is usually applied to problems where pictures are involved. Pictures have really nice property – all features have limited, similarly important values. It could also be the reason why VAE usually smoothes generated samples [5], it exploits spatial dependence, but the model does not have enough power to learn proper borders between objects. The malware detection requires a similar approach – some features are independent, but others are highly dependent. Nevertheless, the features are not similarly distributed.

3 Methodology

3.1 Autoencoders

Autoencoders are artificial neural networks capable of nonlinear space reduction. They are trained almost like classical neural networks for regression, but their objective is the reconstruction of their input. The reconstruction function is any loss function that could be used for regression problems. A usual choice is the mean squared error (MSE) or the mean absolute error (MAE). Their main benefit comes from the introduction of a bottleneck. The bottleneck can be a narrow part of the network or some heavy regularisation. Such a bottleneck is required to have a restricted flow of information. This restriction causes an autoencoder to preserve only the most important patterns in data and drop that kind of information that can be easily deduced or is noisy. An interesting fact is that neural networks with identity activation function and MSE loss converge to the same mapping as the principal component analysis transformation [6].

There exist several kinds of autoencoders. Most commonly, they are used as a dimensionality reduction method, or for denoising – where the input is formed by corrupted image while a crisp image is expected as an output [1, 5].

3.2 Variational Autoencoder (VAE)

Sometimes new data is required to be produced based on a small number of real samples. For these types of problems, autoencoders offer a solution. Bottlenecks significantly reduce chances for overtraining while maintaining appropriate power due to nonlinear space reduction.

One problem remains: if we would like to generate a new sample, we need to know the distribution of codings. For this reason, VAE adds two updates to regular autoencoder that makes codings obey some distribution (usually normal). The first update is to add another term in the model's loss. The Kullback-Leibler (KL) divergence is a measure that relates two distributions μ and ν and is defined by

$$\begin{aligned} D_{\mathrm{KL}}(\mu \parallel \nu) &= \mathrm{H}(\mu,\nu) - \mathrm{H}(\mu) \\ &= -\int_{\mathbb{R}^n} \mu(x) \ln \nu(x) dx + \int_{\mathbb{R}^n} \mu(x) \ln \mu(x) dx \\ &= \int_{\mathbb{R}^n} \mu(x) \ln \left(\frac{\mu(x)}{\nu(x)} \right) dx, \end{aligned}$$

where $H(\mu,\nu)$ is cross-entropy and $H(\mu)$ is entropy. If μ and ν equals, the divergence is 0. Otherwise, it is a positive value.

The model can link small changes in codings with large changes in result, giving an opportunity to overtraining. So the second update is to add uncertainty into codings. In practice, the coding layer of VAE is split into two branches, one for means and one for variances. The second part of the network – decoder then receives samples based on these parameters of the generating distributions. In fact, it is common to use the logarithm of variance since it has a more suitable range. Then a loss based on the KL divergence can be reduced to

$$\mathrm{L}_{\mathrm{VAE}} = \mathrm{L}_{\mathrm{Reconstruction}} - \frac{1}{2N} \sum_{i=1}^{N} \sum_{l=1}^{G} \left(1 + v_{il} - m_{il}^2 - e^{v_{il}}\right)$$

where N is the batch size, G is the coding dimension, v_{ij} is the coding variance and m_{ij} is the coding mean [6]. A scheme of a small VAE is depicted in Figure 1.

3.3 Wasserstein distance

Let $\|\cdot\|$ be an arbitrary norm on $\mathbb{R}^n, n \in \mathbb{N}$ and μ, ν be probability measures on $\mathbb{R}^n$ with finite first moments. For each n-dimensional random vector X, denote $\mathscr{D}(X)$ the distribution of X. Then the Wasserstein distance of order 1, or simply Wasserstein distance (aka Wasserstein measure, Kantorovich-Rubinstein

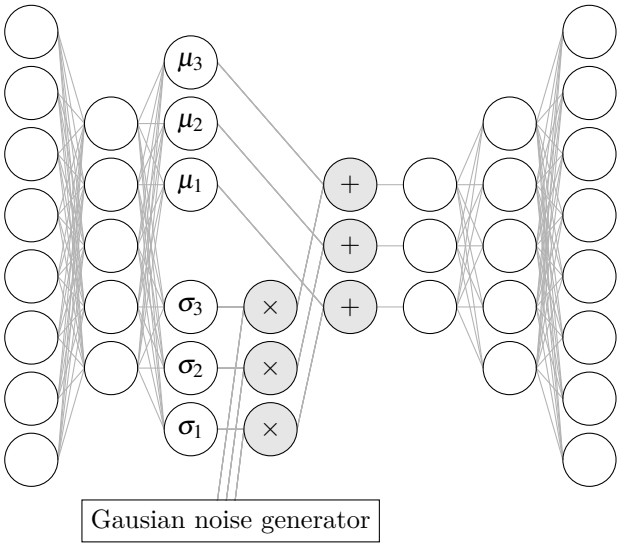

Figure 1: Variational autoencoder. Gray nodes are operations, μ_i, σ_i nodes have linear activation fucntion.

distance, earth mover's distance) between μ and ν corresponding to the norm $\|\cdot\|$ is defined [4, 11]:

$$W_1(\mu,\nu) = W(\mu,\nu) = \\ = \inf\{\mathbb{E}\|X-Y\| \mid \mathscr{D}(X)=\mu \wedge \mathscr{D}(Y)=\nu\}. \quad (1)$$

We decided to use the Wasserstein distance as a method for model comparison. Unlike Kullback-Leibler divergence, it is a metric – it is symmetric and fulfills the triangle inequality. Moreover, it does not require the considered measures to share a probability space. The Wasserstein distance is also scale sensitive $W(c\mu, c\nu) \leq \mathfrak{c}^{\beta} W(\mu,\nu)$ and sum invariant $W(\mu+A,\nu+A) \leq W(\mu,\nu)$ for any $\beta, c > 0$ and $A \in \mathbb{R}^n$ [2]. Therefore, it provides us some intuition about our results.

3.4 Considered kinds of normalization

All included attributes of the data were normalized with 14 different transforming functions, denoted $T_1 - T_{14}$. The definitions of nearly all of them are quite simple, therefore, they are merely listed in Table 1. The only exception is T_5, which performs a power transformation proposed by Yeo and Johnson [12]. It is based on the popular Box-Cox transform [3].

We use the following notations: $a \in \mathbb{R}$ denotes an arbitrary value of a transformed attribute A, whereas $\mathbb{A}$ denotes the set of values of A in the training data $a_1, \ldots, a_p$, $\|\mathbb{A}\|$ denotes the set of absolute values of A in $\|a_1\|, \ldots, \|a_p\|$, $A_q, q \in (0,1)$ denotes the quantile q of a sample of A in the training data, ϕ is the cumulative distribution function of the standard normal distribution $N(0,1)$, and $\mathfrak{e}$ is the Euler constant, on which natural logarithms are based. Because T_6, T_7, T_{13}, T_{14} are based on quantiles prediction, the final transformations are formed by substitute linear interpolations for those quantiles. Due to performance reasons, the quantiles are predicted based on 100000 samples of A.

Table 1: Functions considered as normailizing transformations of data attributes.

Function definition	Mnemonic name
$T_1(a) = \frac{a-\min\mathbb{A}}{\max\mathbb{A}-\min\mathbb{A}}$	min-max
$T_2(a) = \frac{a}{\max\|\mathbb{A}\|}$	max-abs
$T_3(a) = \frac{a-\sum_{j=1}^{p} a_j}{\sqrt{\sum_{i=1}^{p}(a_i - \frac{1}{p}\sum_{j=1}^{p} a_j)^2}}$	standardize
$T_4(a) = \frac{a-A_{0.5}}{A_{0.75}-A_{0.25}}$	robust
$T_5(a) = \psi(a, \hat{\lambda})$ with ψ and $\hat{\lambda}$ defined in (2)–(3)	power
$T_6(a) = q$ such that $A_q = a$	perc-uniform
$T_7(a) = \phi^{-1}(T_6(a))$	perc-normal
$T_8(a) = T_1(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-min-max
$T_9(a) = T_2(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-max-abs
$T_{10}(a) = T_3(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-standardize
$T_{11}(a) = T_4(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-robust
$T_{12}(a) = T_5(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-power
$T_{13}(a) = T_6(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-perc-uniform
$T_{14}(a) = T_7(\log(a+\mathfrak{e}-\min\mathbb{A}))$	log-perc-normal

This function is defined in four steps:

I. A parametrized version ψ of the intended transformation is defined, which depends on a parameter $\lambda \in \mathbb{R}$:

$$\psi(a,\lambda) = \begin{cases} \frac{(a+1)^{\lambda}-1}{\lambda} & \text{if } a \in \mathbb{R}_{\geq 0}, \lambda \neq 0, \\ \ln(a+1) & \text{if } a \in \mathbb{R}_{\geq 0}, \lambda = 0, \\ -\frac{(-x+1)^{2-\lambda}-1}{2-\lambda} & \text{if } a \in \mathbb{R}_{<0}, \lambda \neq 2, \\ -\ln(-x+1) & \text{if } a \in \mathbb{R}_{<0}, \lambda = 2. \end{cases} \quad (2)$$

II. It is assumed that for some range of values of the parameter λ, the value of $\psi(a,\lambda)$ is a random variable with the distribution $N(\mu,\sigma^2)$ for some $\mu \in \mathbb{R}, \sigma \in \mathbb{R}_{>0}$. Consequently, the log-likelihood of the parameters (λ,μ,σ^2) with respect to training data is

$$\ell(\lambda,\mu,\sigma^2; a_1,\ldots,a_p) = \\ = -\frac{p}{2}\log(2\pi) - \frac{p}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{p}(\psi(a_i,\lambda)-\mu)^2 \\ + (\lambda-1)\sum_{i=1}^{p}\operatorname{sign}(a_i)\log(\|a_i\|+1).$$

III. The parameters μ and σ^2 are for each $\lambda \in \mathbb{R}$ estimated, respectively, with the following estimates $\hat{\mu}(\lambda)$ and $\hat{\sigma}^2(\lambda)$:

$$\hat{\mu}(\lambda) = \frac{1}{p}\sum_{i=1}^{p} \psi(a_i, \lambda),$$
$$\hat{\sigma}^2(\lambda) = \frac{1}{p}\sum_{i=1}^{p} (\psi(a_i, \lambda) - \hat{\mu}(\lambda))^2.$$

IV. For the parameter λ, the maximum likelihood estimate is used, after replacing the other two parameters by their estimates:

$$\hat{\lambda} = \max_{\lambda \in \mathbb{R}} \ell(\lambda, \hat{\mu}(\lambda), \hat{\sigma}^2(\lambda); a_1, \ldots, a_p) =$$
$$\max_{\lambda \in \mathbb{R}} (\lambda - 1) \sum_{i=1}^{p} \operatorname{sign}(a_i) \log(\|a_i\| + 1) - \frac{p}{2} \log \sigma^2(\lambda). \tag{3}$$

We cannot measure the Wasserstein distance in transformed space because there is no clear way of comparing the results between transformations. It is not a problem for most of the transformations we use, but the perc-uniform and perc-normal are slightly problematic. We have to use a linear interpolation on a sequence of quantiles in order to approximate the initial dataset. For this reason, we expect to have some increase in the Wasserstein distance on perc variants since it is nonzero value even if we compare original dataset to its forward and then backward transformed version.

4 Experimental Evaluation

4.1 Available Malware Data

We use real-word anonymized data, which feature malware and clean software in several categories, but we consider only two by merging some of them. Anti-malware software producers must protect their know-how by providing only data which were anonymized. For us, the anonymization process is unknown. It essentially hides the meaning of features in provided datasets by stripping it in the documentation and applying unknown functions on those columns. The feature space is very complex, there are 540 features with various distributions. This makes particularly difficult to choose the correct data scaling. In Figure 2, several kinds of features are differentiated:

- Binary feature
- Gaussian feature: both absolute skewness and excess kurtosis are less than 2
- Highly skewed feature: skewness > 30
- Almost constant feature: more than 99.9 % values are identical
- Other unknown distributions

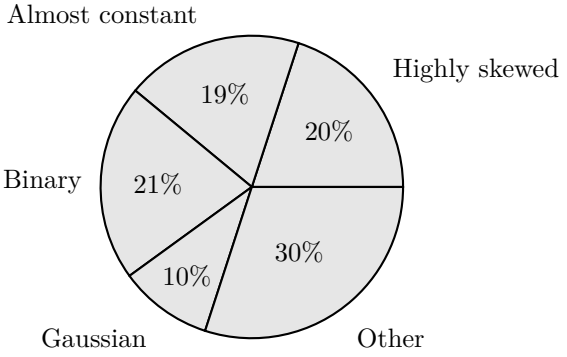

Figure 2: Distribution in the feature space.

4.2 Experiments with Different Normalizations

Our dataset has many outliers. We were interested in how outliers effect generator. Therefore, we compared two common loss functions used for regression, both combined with the kinds of normalization considered in Subsection 3.4. The first was the mean square error (MSE) and the second was the mean absolute error (MAE). Both are tested on a rather large network with 7 layers $[541, 306, 173, 98, 56, 32, 18, 10]$, with ELU as activation function and batch normalisation on every layer except the coding and output one. The results can be seen in Figure 4 It looks like the best normalisation for our data is the power transform, but the differences between different kinds of normalizations are, in general, small. It transforms data to be more normal. Both perc-uniform and perc-normal performed excellently. It seems that small distortions created by linear interpolations of quantiles do not deteriorate result.

It is better to use raw data than robust, min-max and max-abs types of normalisations, especially if MSE is employed. We observe that some of the features consist of a small-valued majority with some huge values. Min-max and max-abs transformations could make this range very small, demanding high precision on regression for the smaller ones. On the other hand, the robust transformation could make the range larger because quantities could be evaluated only on the small values – artificially increasing existing variance.

It does not seem that the log variants have any positive effect, but we saw a slight increase in min-max and max-abs transformations. This is expected because logarithm reduces ranges of values. Robust transformation has a substantial incompatibility with logarithm transformation because log-robust transformation operates on logarithmic space. After exponentiation, the errors could be much larger. However, this explanation fails if we consider that log-perc transformations performed almost equally.

MAE is generally better. MSE is equal to MAE in perc-uniform, perc-normal, log-perc-uniform, log-perc-normal, power, log-power.

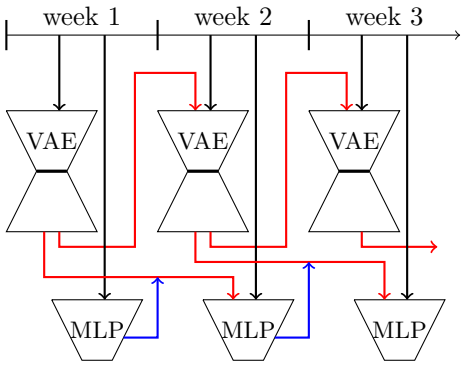

Figure 3: Training data paths for VAEs and MLPs for each week. Red indicates generated data, blue adds label classifications to features.

We also tried to investigate how an increase in coding size affect the resulting performance. It seems there is no difference, 10 as a coding dimension (predicted normal distributions of 2 parameters) is sufficient no matter which normalisation was chosen. The results can be seen in Figure 5.

Figure 6 shows expected behaviour. The VAE has trouble expressing features with higher variance. It is evident, especially on the highest tertile, where differences between distributions are almost non-existent.

A next view on the data reveals something interesting – there is a great difference between the data with different signs of skew of a particular feature. Negative skew was much harder to imitate while positive skew was easier to reproduce than in the other two middle quartiles. This could be a property of dataset features or under-training. It seems that max-abs and min-max normalizations have a slightly different response.

4.3 Experiments with Different MLPs

We were interested in the ability of generative models to capture important information (e.g. vanishing or strange behaviour) and ignore the noise (waiting for user input). As the next experiment, we prepared a balanced dataset extended by a feature reporting true benign and malware labels. Firstly, we let the generative model learn on the first part of data without true labels. Then, every other part of the data was augmented by classified features generated by the previous generator and classified by the previous classifier. This process is depicted in Figure 3.

We prepared two results of Bayesian optimisation of MLPs hyperparameters. The first, MLP_1 is the result of mid-optimisation and MLP_2 is the final result. The hyperparameters are in Tables **??** and **??**. Even thought the MLP_1 is significantly worse than MLP_2 in 94% of weeks, when VAE was added, it became significantly better on 18% of weeks while MLP_2 on none them.

In Figure 8, unusual behaviour of the compared methods can be seen. The VAE with the MLP_1 is significantly better than the VAE with the MLP_2 for the weeks 1-71. This is interesting because a separate MLP_1 trained on a small amount of samples is generally worse than MLP_2.

Several possibilities explain such behaviour. Firstly, the hyperparameters of MLP_2 can be simply overtrained on training samples. Instead of learning about patterns, the model may learn small mistakes made by VAE and try to infer based on them. Secondly, MLP_1 may lack some form of regularization.

5 Conclusion

This paper investigated the application of variational autoencoders to online learning from malware detection data. To this end, it employed a large real-world dataset of anonymized high-dimensional data. We prepared a hybrid model that consists of multiple pairs of a generator and a classifier. The basic idea of our approach is to make the generators learn the history and later use it for data augmentation. In addition, the influence of different kinds of data normalization on the VAE perfomance has been investigated. For the assessment of VAE performance, accuracy has been complemented with the Wasserstein distance.

The experiments have proven our expectation that for highly heterogeneous data, normalization is relevant. However, the most basic transformations like min-max and max-abs were worse than using the raw value without any transformation. The experiments have also shown that variational autoencoders can be used for data augmentation in MLP-based classification. However, one should be aware of the hyperparameters determining the size of the classifying MLP. If it is too large, generated noise produced by VAE could be detrimental instead.

Acknowledgement

The research reported in this paper has been supported by the Czech Science Foundation (GAČR) grant 18-18080S. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

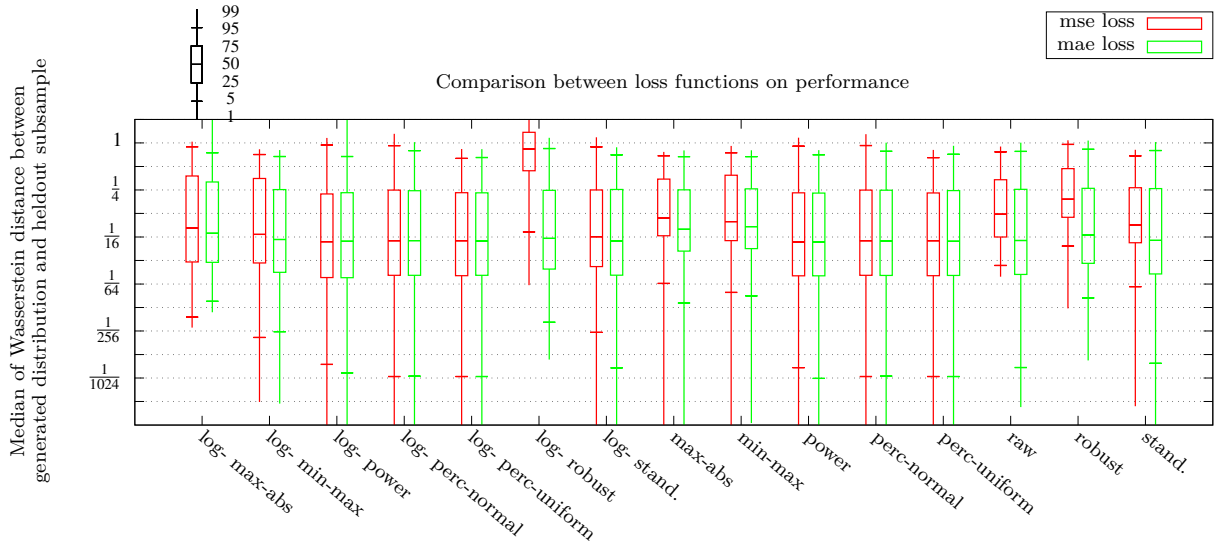

Figure 4: The graph shows reactions of the VAE on the loss function given a dataset normalization.

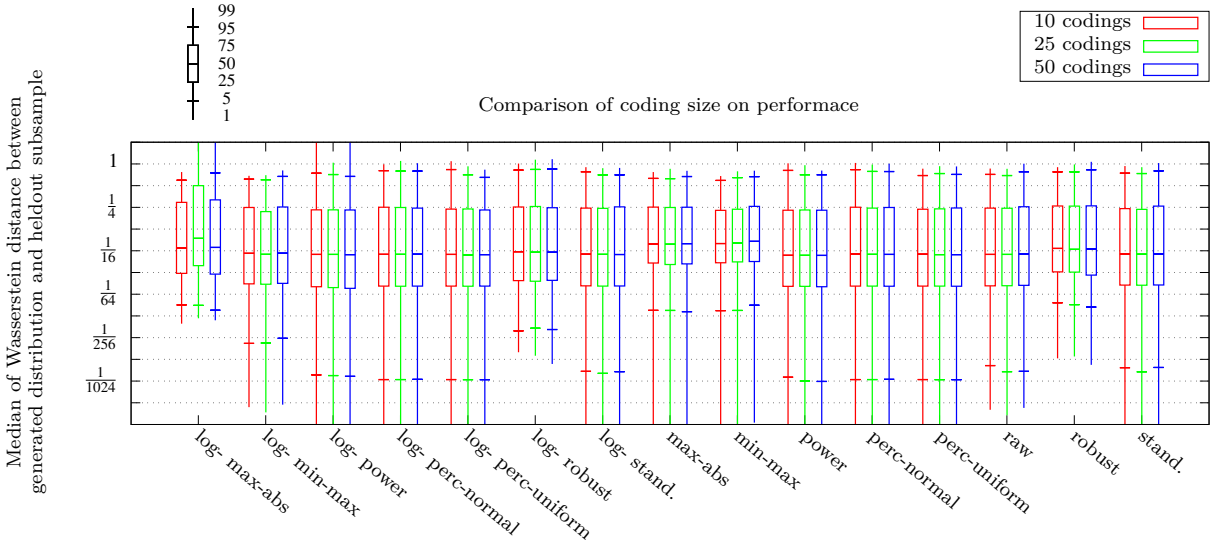

Figure 5: The graph shows reactions of the VAE on the increase in coding size given a dataset normalization.

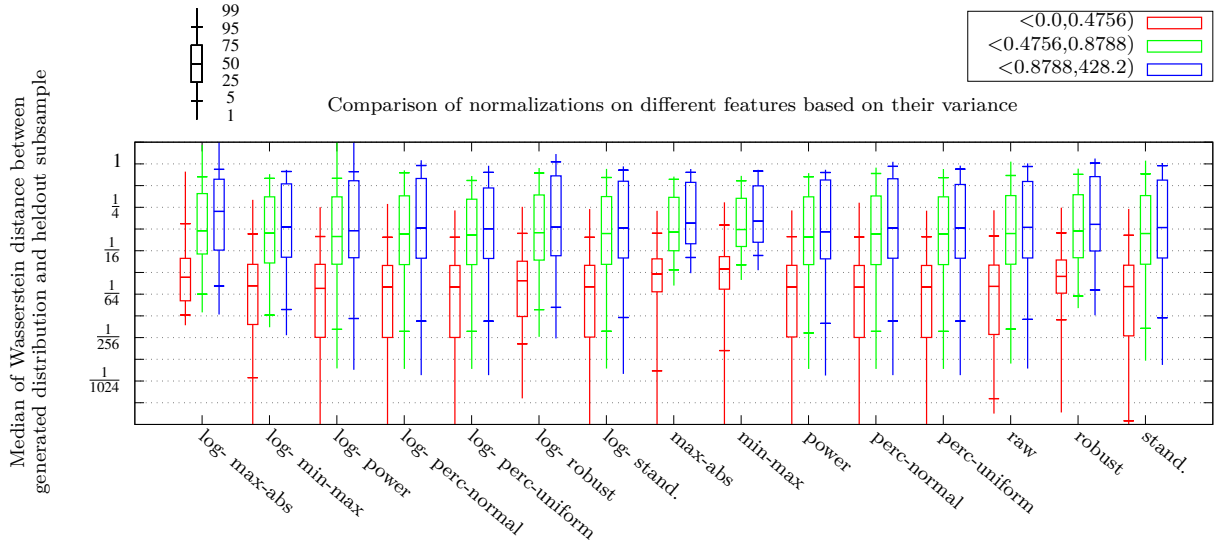

Figure 6: The graph shows how the VAE model handle features with different variances given a dataset normalization.

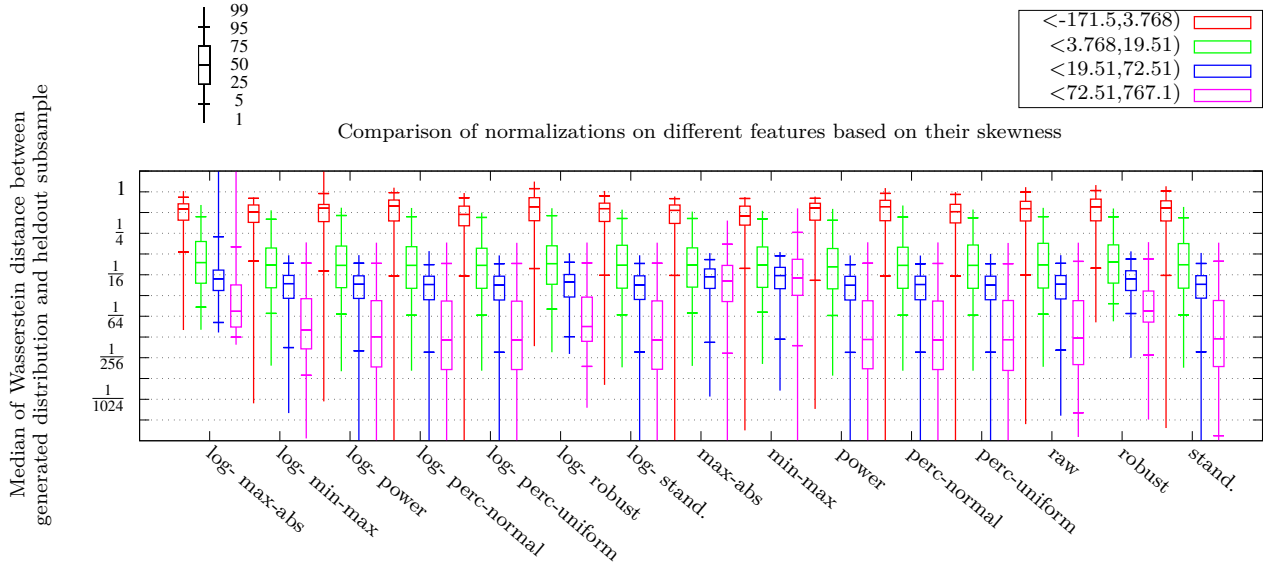

Figure 7: The graph shows how the VAE model handle features with different skew given a dataset normalization.

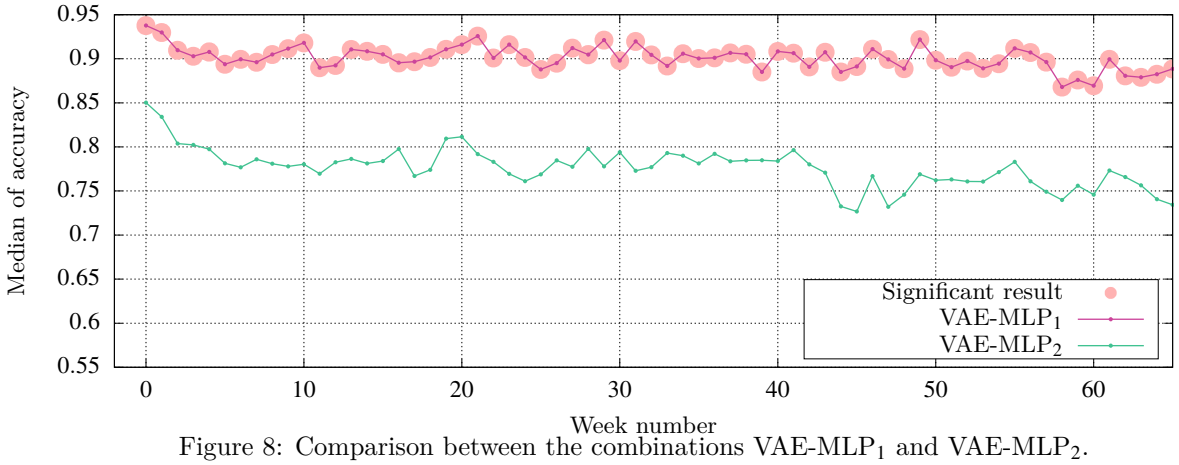

Figure 8: Comparison between the combinations VAE-MLP$_1$ and VAE-MLP$_2$.

Table 2: Results of MLP$_1$ and MLP$_2$ hyperparameter optimization using GPyOpt library.

Name	Possibilities	Selected values	
		MLP$_1$	MLP$_2$
Learning rate	0.0001-0.01	0.0002	0.00763
Batch norm.	yes/no	yes	yes
Dropout	0-0.7	0	0.22
Gaussian noise	0-1.0	0	0.795
Layers	1-1-1-2 up to 400-400-400-400-2	69-32-30-11-2	354-322-316-305-2
Activation	ELU, SELU, softplus, softsign, ReLU, tanh, sigmoid, leaky ReLU, PReLU	tanh	ReLU
Minibatch size	10-1000	410	730
L1 regularization	0-0.1	0.00016	0.01
L2 regularization	0-0.1	0.09229	0.0998
Data scaling	standard, robust min-max	min-max	standard

References

[1] G. Aurélien. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly, 2 edition, 2019.

[2] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos. The cramer distance as a solution to biased wasserstein gradients. *CoRR*, abs/1705.10743, 2017.

[3] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[4] P. M. Esfahani and D. Kuhm. Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming, Series A*, 171:115–166, 2018.

[5] D. Foster. *Generative deep learning: teaching machines to paint, write, compose, and play*. O'Reilly, 1 edition, 2019.

[6] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[7] A. Narayanan, L. Yang, L. Chen, and L. Jinliang. Adaptive and scalable Android malware detection through online learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2484–2491, July 2016.

[8] B. Rashidi, C. Fung, and E. Bertino. Android Resource Usage Risk Assessment using Hidden Markov Model and Online Learning. *Computers & Security*, 65, November 2016.

[9] B. Rashidi, C. Fung, and E. Bertino. Android malicious application detection using support vector machine and active learning. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9, November 2017.

[10] J. Tumpach, M. Krčál, and M. Holeňa. Deep networks in online malware detection. In *ITAT*, pages 90–98, 2019.

[11] C. Villani. *Optimal Transport, Old and New*. Springer, 2006.

[12] I. K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87:954–959, 2000.