




# Word Embeddings for Fake Malware Generation

Quang Duy Tran and Fabio Di Troia<sup>(✉)</sup> 

San José State University, San Jose, USA  
fabio.ditroia@sjsu.edu

**Abstract.** Signature and anomaly-based techniques are the fundamental methods to detect malware. However, in recent years this type of threat has advanced to become more complex and sophisticated, making these techniques less effective. For this reason, researchers have resorted to state-of-the-art machine learning techniques to combat the threat of information security. Nevertheless, despite the integration of the machine learning models, there is still a shortage of data in training that prevents these models from performing at their peak. In the past, generative models have been found to be highly effective at generating image-like data that are similar to the actual data distribution. In this paper, we leverage the knowledge of generative modeling on opcode sequences and aim to generate malware samples by taking advantage of the contextualized embeddings from BERT. We obtained promising results when differentiating between real and generated samples. We observe that generated malware has such similar characteristics to actual malware that the classifiers are having difficulty in distinguishing between the two, in which the classifiers falsely identify the generated malware as actual malware almost 90% of the time.

**Keywords:** BERT · GAN · Malware · Malware detection · Word embedding

## 1 Introduction

The term malware refers to software that is created with the intention of causing damage to computer data [2]. According to Statista, a total of 5.6 billion malware attacks took place worldwide in 2020 [15]. These attacks target many small and large industries, such as finance, transportation, healthcare, manufacturing, or professional, which can cause immeasurable damage. For this reason, malware prevention has become a vital part of information security.

Recently, machine learning approaches have been utilized in the malware detection area to combat these threats. The common method of training intelligent models is by collecting the malware characteristics such as opcode sequences, API calls, and bytes vectors, among others [3]. Despite the promising results from machine learning techniques, there are still significant obstacles

to overcome, such as adversarial machine learning to deceive machine learning models [17], malware code obfuscation [5], and the shortage of publicly available training datasets [28].

In this research, we build realistic fake malware samples from seven distinct malware families by applying Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP). To build such realistic samples of a malware infection, it is crucial to identify the distinctive features of each malware family [33]. In general, malware samples belonging to the same family share many characteristics with one another, yet they differ from those belonging to other families. These unique characteristics of a malware sample can be quantified by word embeddings, which can be generated by many Natural Language Processing (NLP) models, including BERT. Thus, the focus of this study is on the effectiveness of word embeddings developed in the context of creating malware samples. We employ a variety of machine learning classification techniques, including One-Class Support Vector Machine (OCSVM), Isolation Forest, and Local Outlier Factor, to assess this effectiveness by differentiating between real and generated samples.

The rest of the paper is structured as follows. Section 2 reviews earlier and related work. In addition, it provides a brief overview of the machine learning techniques and concepts employed in this study. Section 3 explains our methodology, including our malware creation pipeline, and the training and evaluation procedure. Section 4 discusses the actual implementation and our experimental setup. Section 5 contains the analysis and results of our experiments. Finally, in Sect. 6, we present the conclusions and future directions of this paper.

## 2 Background

In this section, we discuss related work in applying Generative Adversarial Networks (GANs) to generate malware samples. Also, we briefly introduce the machine learning techniques used in this research.

### 2.1 Selective Survey of Related Work

Creating images from malware executable files and utilizing them to perform malware detection and classification is a current trend in modern malware research. This enables the use of image-analysis techniques, as well as the operations of strong deep neural networks that function well with images. Many researchers utilized malware images to produce malware samples for generative models, since it allows them to add more samples to the pool and even to perform data augmentation on real data. For instance, in [21] the authors applied Variational Auto Encoder (VAE) and GANs to expand the training dataset using malware as images, resulting in 2% and 6% increase in accuracy, respectively. Another similar study [29] implemented GAN for the same purpose and obtained a 6% increase in accuracy when trained ResNet-18 model on malware data.

Although treating malware as images and perform augmentation has gained popularity, we should not ignore the drawback of this technique, which requires huge computational resources. Additionally, training and testing deep convolutional networks can be computational expensive and time-consuming. Understanding the disadvantages, the authors in [31] proposed another approach based on generating mnemonic opcode sequences by applying and comparing Hidden Markov Models and three different GAN architectures. They were able to fool the classifiers in distinguishing between real and generated malware with a 76% detection/accuracy score when using WGAN-GP. Their main goal, however, was to experiment with various generative modeling techniques when building mnemonic opcode sequences to represent fake malware. In this work, we push further the previous study by introducing word embedding generation. The main difference between our study and [31] is that we utilize word embedding technique to generate realistic malware. Even though our approach is very similar to [31], we will not compare our results to theirs because our evaluation method is different.

Natural Language Processing (NLP) techniques can extract rich information from sentences in a language, known as word embeddings. These embeddings are able, for instance, to capture the meaning of a phrase, construct sentences with similar meaning, or fill in the blanks within a sentence. NLP models extract information about a word’s relationship to every other word in a phrase. The model then clusters words with similar meanings together in a higher dimensional space. This information assists NLP models in performing classification and prediction tasks. In the realm of malicious malware, the models are utilized to build embeddings for malware samples. For example, a research [25] was conducted on detecting malware samples by applying NLP to mnemonic opcode sequences. The results, derived by utilizing word embeddings generated by Word2Vec, HMM2Vec, ELMo, and BERT, prove that NLP based models can extract rich features that assist with classification accuracy. In particular, word embeddings generated by BERT consistently achieved a superior classification accuracy compared to other NLP techniques, with an accuracy of around 96%.

We see that there is a gap in the literature when it comes to utilizing word embeddings to assist GAN to generate realistic malware. As a result, we conduct a study on this subject by introducing the usage of BERT in generative modeling. In summary, we extract mnemonic opcodes from malware files, apply BERT transformer to obtain embeddings, and train generative models to generate malware embeddings.

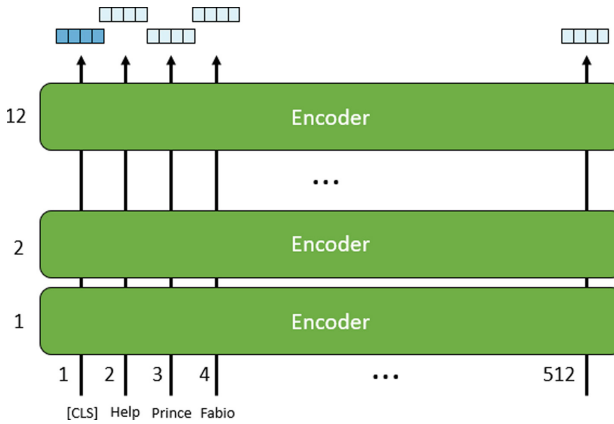
## 2.2 Machine Learning Techniques

In this section, we discuss the key elements of this paper, that is, the generative and NLP models. The generative model that we implemented is Wasserstein GAN with Gradient Penalty (WGAN-GP), while the NLP model is BERT.

**Wasserstein GAN with Gradient Penalty.** Ishaan Gulrajani et al. introduced WGAN-GP in 2017 [32]. The fundamental goal of this design is to surpass

WGAN’s disadvantage by applying gradient norm penalty with Wasserstein loss formulation to achieve Lipschitz continuity. The authors provide Corollary 1, which states that the optimum critic in WGAN has a gradient norm of 1 and is 1-Lipschitz continuous. Taking advantage of this, a “penalty” is placed on the critic if the norm of its gradient deviates from the value 1. WGAN-GP training method is similar to the WGAN algorithm, with the exception of the weight clipping and the inclusion of the gradient penalty [32]. More details about WGAN can be found in [26].

**BERT.** Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based NLP model that has shown to be successful in difficult language-based tasks such as masked word prediction and sentiment classification. The model is capable of creating contextualized word embedding by taking the context in which the individual words are used into account. BERT model finds word relationships via attention, a method that aids in the retention of long-term dependencies in sentences of up to 512 words, as shown in Fig. 1. In general, a normal language phrase does not exceed 512 words, but the opcode sequences in a malware sample can surpass this limit. In our testing, the first 400 opcodes from each malware file were adequate to obtain satisfactory outcomes while using less computational resources. Further details on the architecture and attention mechanism of BERT can be found in [16], while an analysis of its attention heads is covered in [7].



**Fig. 1.** Trained BERT components

**DistilBERT.** Despite the fact that BERT has grown more common in Natural Language Processing, running these huge pre-trained models with tight computational training or inference budgets remains difficult. DistilBERT, a computationally efficient and lightweight version of BERT, was proposed to overcome this challenge. DistilBERT model was constructed using the knowledge distillation

during the pre-training phase to shrink a BERT model by 40%, while preserving 97% of its language comprehension skills and being 60% faster [4]. This model, similar to its larger counterparts, is also capable of being fine-tuned with good performances on a wide range of tasks. Seeing the advantages the model offers, we utilized DistilBERT in our study to create contextualized malware embeddings. The DistilBERT model that we used is hosted and open sourced by HuggingFace team [6], and was pre-trained on the English language. In this study, the model was neither trained nor fine-tuned on malware samples. More details about the model can be found in [4].

### 3 Methodology

This section provide details on the dataset, as well as the training and evaluation procedure we followed in our experiments.

#### 3.1 Dataset

The dataset that we used for all of our experiments consists of seven malware families with more than 1000 samples (shown in Table 1).

**Table 1.** Dataset summary

Malware family	Type	Samples
WinWebSec	Rogue	4360
VBInject	Worm	2694
Zbot	Information Stealer	2136
Renos	Trojan Downloader	1568
OnLineGames	Password stealer	1513
BHO	Trojan	1405
Zeroaccess	Corrupting Devices	1305

To begin with, we looked into the Malicia dataset [18], which has over 50 malware families, and selected Winwebsec, Zbot, and Zeroaccess since they have over 1000 samples each. The remaining four families were obtained from VirusShare [30]. There are over 120,000 malware executables in this 100 Giga-bytes dataset, from which four extra families were selected, namely, VBInject, Renos, OnLineGames, and BHO. These families are in top 5 most sample counts, where each has more than 1300 samples. Winwebsec and Zbot are also provided in the VirusShare dataset. However, we use the ones from Malicia dataset because they have almost double the sample size.

We utilized *objdump*, *sed*, and *cut*, the command-line utilities included in the GNU Binary Utilities package for Unix-like operating systems. Using *objdump*,

executable files can be disassembled into Assembly code, thus allowing them to be extracted as mnemonic opcodes. Then, *sed* and *cut* can be used to filter and transform text to remove all the excessive information, such as registers or addresses.

Here is a short description of our selected malware families.

**Winwebsec:** a trojan that creates fictitious problems and claims to have solutions that are authentic anti-virus software [8].

**Zbot:** a trojan that breaches into Microsoft Window computers and steals confidential information [9].

**Zeroaccess:** a trojan that infects Window computers and exploits them for malicious purposes such as corrupting devices [10].

**BHO:** performs a wide range of harmful behaviors as directed by an attacker [14]

**OnLineGames:** steals users' login credentials and records keystroke activity [12]

**Renos:** pretends that the machine is infected with malware and demands money to remove the non-existent spyware [13]

**VBInject:** packed malware that obfuscates its content to conceal other malware and itself from detection [11]

### 3.2 Training Procedure

Figure 2 depicts an example of our WGAN-GP training process. Tokenizers are applied to the original malware dataset before turning into BERT inputs that generate word embeddings. These embeddings are then directly used to train the generative model. After training, WGAN-GP is able to generate fake malware embeddings as outputs. We rinse and repeat this entire process through all malware families.

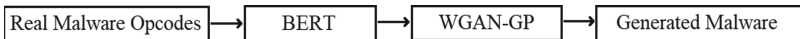
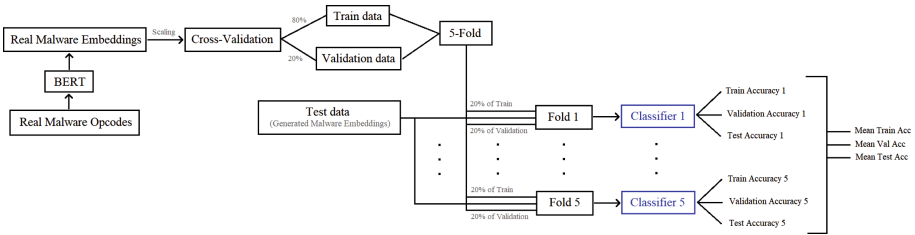


Fig. 2. WGAN-GP training process

### 3.3 Evaluation Procedure

Figure 3 shows an illustration of our WGAN-GP evaluation procedure. The initial set up is partially the same as the training procedure, where we generated real malware embeddings from malware opcodes using DistilBERT model. Following, we apply 5-fold cross-validation and split the data into 80% training and 20% validation set. Test data contains malware embeddings generated by WGAN-GP, and is split into five subsets to participate in 5-fold cross-validation. After, we construct three new classifiers (One-Class SVM, Isolation Forest, and Local Outlier Factor) inside each fold to train and assess on the validation and

test dataset. At the end of each fold, we obtain train, validation, and test accuracy. Next, we calculate the average of the 5 folds as the final results for that WGAN-GP model. We cycle over all generative models that were trained on that same malware family, and compare their results to determine which model has the lowest test accuracy score. The lowest score indicates that the model was the most successful in generating fake samples able to confuse the detection algorithms. Such model is, thus, selected as the best generative model for that particular malware family. This same process is repeated for all malware families.



**Fig. 3.** WGAN-GP evaluation process

## 4 Implementation

In this section, we discuss the techniques we applied to extract embeddings using BERT model. Moreover, the parameters used for the generative and classification models are also provided.

### 4.1 Feature Extraction

The classification tokens (CLS) gather information about the entire sentence and are used to express sentence-level classification results. In the instance of a malware sample, the CLS token collects all of the sample’s information. This information can be used to aid the learning of WGAN-GP. Therefore, among the 768 hidden units of BERT, the first column (that represents the CLS tokens) is selected. Furthermore, leveraging image scaling from image processing, we applied the same principle to BERT embeddings to help ease the learning of WGAN-GP. The embeddings were also scaled to fit into the range of  $-1$  and  $+1$ . This technique simplifies BERT embeddings, which allows the generative models to convert faster.

## 4.2 WGAN with Gradient Penalty

In our study, we took WGAN-GP into consideration because the network has been shown to perform well in [31]. We were influenced by the study of the authors in [31], and used the same parameters in our experiments. Adam optimizer was used with the following parameters:

$$Adam(lr = 0.0001, \beta_1 = 0.5, \beta_2 = 0.9)$$

Each WGAN-GP model was trained for 100,000 epochs. The critic network consists of three hidden Conv1D layers with 64, 128, and 256 filters and a kernel size of 3. Similarly, a kernel size of 3 and three Conv1D layers with 64, 32, and 16 filters were utilized in the generator network. LeakyReLU is the activation functions for the hidden layers of Conv1D.

In the generator, the output layer consists of a fully connected Dense layer with 768 neurons. The reason to use exactly 768 neurons is to match the output size of BERT. The activation function for the generator is TanH, while there is none for the critic network. The authors in [31] decided to not implement neither Batch Normalization nor Layer Normalization in the critic network. But in the generator, Batch Normalization is still applied.

The penalty coefficient,  $\lambda$ , is set to 10. The parameter “n\_critic”, which represents the number of critic iterations per generator iteration, is set to 100. In other words, for each epoch, the generator was only updated after training the critic for 100 iterations. Table 2 shows all the parameters and their values used in the generator and critic.

**Table 2.** Generator and Critic parameters

	Parameter	Value
Generator	activation	TanH
	kernel_size	3
	Conv1D_layer_filter_1	64
	Conv1D_layer_filter_2	32
	Conv1D_layer_filter_3	16
	Conv1D_padding	same
	BatchNorm_momentum	0.8
Critic	activation	LeakyReLU
	kernel_size	3
	Conv1D_layer_filter_1	64
	Conv1D_layer_filter_2	128
	Conv1D_layer_filter_3	256
	Conv1D_padding	same



### 4.3 Evaluation Implementation

Because GAN are most commonly used in the image domain, the two popular metrics Inception Score [22] and Fréchet Inception Distance (FID) [27] are used to evaluate the quality of the generated images. In addition, generated images are saved every few hundred epochs, such as 500, before being examined visually. However, our dataset consists of opcode sequences, which are impossible to be inspected visually. Hence, we applied a different metrics to evaluate GAN performance. Reading about the two scores, we realized that the similarity between them is that they are calculated using the inception-v3 model. Moreover, inception-v3 was trained on more than a million images from the ImageNet database, and attained a greater than 78.1% accuracy on the same dataset [1]. The key point here is that, when evaluating against GAN, the inception-v3 network has neither seen nor learned about the generated images. Therefore, we decided to not include generated data, but only malware data, into our training set. We then use three classification models, that is, One-Class SVM (OCSVM) [23], Isolation Forest [20], and Local Outlier Factor (LoF) [24], which are based on the idea of anomaly detection, to evaluate GAN’s performance.

Anomaly detection can be branched off into outlier and novelty detection. To decide which one to use, we have to look into the difference between the two. Outlier detection is used when there are outliers in the training data, which are observations that are different from the rest of the data [19]. On the other hands, novelty detection is used when the training data is not contaminated with outliers, and we are interested in determining if a new observation is an outlier [19]. Since our train data only contains malware opcodes, novelty detection is selected in our evaluation. Our test dataset, which only contains generated data, will be considered as new observations to be determined if it is anomaly.

In our study, we saved the generative model at every 500 epochs. We then generated fake samples from all saved generative models, and classified if they are outliers or not using the three classification models. Afterwards, we tuned all three classifiers using sklearn GridSearchCV [34] on each malware family to achieve the highest train and validation accuracy. Upon training on the real malware data using 5-fold cross validation, these classifiers will be evaluated against the generated data. Accuracy score is then computed to see how similar the generated data is when compared to the real malware data. Lastly, we compared the score across all our generative models and pick out the best model. Note that our goal is to achieve as low accuracy score as possible. High accuracy shows that the three classifiers classified generated data as outliers, which is not similar to real malware data. And vice versa, low accuracy means the classifiers classified generated samples as inliers, which is similar to real malware data. Table 3, 4 and 5 below are the summary of the tuned parameters we used for the classifiers.

**Table 3.** One-class SVM parameters

Malware family	Parameters		
	nu	kernel	gamma
Winwebsec	0.5	sigmoid	0.1
Zbot	0.01	sigmoid	0.001
Zeroaccess	0.01	sigmoid	0.001
VBIInject	0.01	poly	0.3
BHO	0.01	rbf	0.3
Renos	0.01	rbf	0.01
OnLineGames	0.01	poly	0.3

**Table 4.** Isolation forest parameters

Malware family	Parameters			
	contamination	max_samples	n_estimators	bootstrap
Winwebsec	0.01	auto	1	False
Zbot	0.01	auto	1	True
Zeroaccess	0.01	auto	1	True
VBIInject	0.01	auto	1	True
BHO	0.01	0.5	1	True
Renos	0.01	auto	0.8	True
OnLineGames	0.01	auto	1	True

**Table 5.** Local outlier factor parameters

Malware family	Parameters					
	p	contamination	leaf_size	n_neighbors	algorithm	novelty
Winwebsec	2	0.01	2	50	auto	True
Zbot	1	0.01	2	30	auto	True
Zeroaccess	1	0.01	2	30	auto	True
VBIInject	1	0.01	2	5	auto	True
BHO	1	0.01	2	30	auto	True
Renos	1	0.01	2	5	auto	True
OnLineGames	2	0.01	2	5	auto	True

## 5 Results

This section provides the classification results of our experiments followed by a detailed analysis.

### 5.1 Evaluation Score

The author in [32] suggested that the critic’s loss, which is used to assess WGAN-GP’s performance, should start at a high negative value and then converge to zero. The generator’s loss, on the other hand, can fluctuate and, hence, is not intuitive. Therefore, we start by looking at the loss curves and then the classification results.

The critic loss curves for seven families presented a similar pattern to the one in [31]. In our experiments, they all started at around  $-10$ , then quickly spiked up to around  $-27$  during the first 50 epochs, and slowly converted to around  $-0.19$  after 10,000 epochs. In other words, our model was trained properly when it exhibited this behavior. Figure 4 below shows the ranking (from worst to best) of critic loss on all malware families at the end of training.

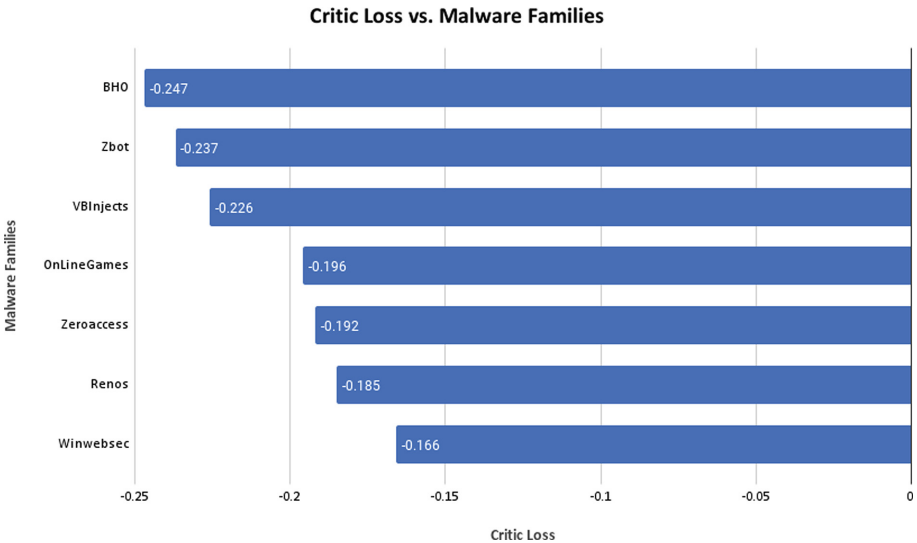


Fig. 4. Ranking of critic loss on malware families

The best test accuracy scores were selected independently from all generative models, the seven malware families, and the three types of classifiers. Our parameters for each classifier were the same as those discussed in Table 3, 4, and 5. The average training accuracy score across seven families was roughly 99.1%. The average validation accuracy score was roughly the same as training accuracy, about 99%. However, the test accuracy score is consistently low, achieved the highest score of 38.8% with BHO, and the lowest score of 0.00%

with Renos. The average of the test score across all families was 15.6%. One-Class SVM showed to perform better than Isolation Forest and Local Outlier Factor in terms of classifying between the real and generated malware data. Most of the test accuracy scores obtained by OCSVM were over 0.00%, except for Renos family.

The Isolation Forest Classifier obtained an average training accuracy of 99.4%. The average validation accuracy score was around 99.4%, which was similar to the training accuracy. However, unlike OCSVM, Isolation Forest obtained lower average test accuracy score (about 2.5%). There are a few more families having the test accuracy scores closer to zero compared to OCSVM.

The average of training and validation accuracy scores of Local Outlier Factor are 99.4% and 98.9%, respectively. There are more families that fooled the classifier that could not distinguish between real and generated malware data. BHO, however, obtained the highest test accuracy in all three classifiers (38.8% for OCSVM, 14.4% for Isolation Forest, and 75.7% for Local Outlier Factor). The complexity and lack of samples for the BHO family could be a big contributing factor to this result. Having the second least amount of samples, 1405 data points, WGAN-GP was not able to adequately learn the distribution of this family. This is further supported by having the highest negative critic loss ( $-0.247$ ) in the training procedure compared to other families. We computed the average accuracy from all test score, and obtained a 12.12% average accuracy. Table 6 summarized the comparison of the accuracy score between each family and classifier. We see that the majority of the families were recreated correctly by our BERT-GAN approach, with the exception of the BHO family for which a considerable number of samples were still classified as fake by LoF and OCSVM.

## 5.2 Further Analysis

To understand the importance of the accuracy score, we should take a look at its formula:

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

In our case, *Positive* represents real malware and *Negative* represents generated malware. Since our test dataset only contains generated malware, *True Positive* is always 0, that is, cases that are correctly classified as real malware. Similarly, *False Negative* is always 0. Hence, the formula simplified as follow:

$$Accuracy = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

The new formula measures the true negative rate, which is the classifiers' ability to predict a true negative of each category available. Thus, when we achieved 12.12% average accuracy, this tells us that the classifiers are only able to correctly identify generated malware 12% of the time. In other words, if we run the evaluation to classify between real and generated malware 100 times,

**Table 6.** WGAN with gradient penalty scores.

Malware Family	One Class SVM			Isolation Forest			Local Outlier Factor		
	Train	Val	Test	Train	Val	Test	Train	Val	Test
Winwebsec	0.995	0.995	0.044	0.998	0.998	0.0	0.993	0.992	0.0
Zbot	0.991	0.990	0.125	0.992	0.996	0.006	0.992	0.991	0.081
Zeroaccess	0.989	0.989	0.163	0.998	0.996	0.0	0.992	0.989	0.0
BHO	0.983	0.981	0.388	0.992	0.992	0.144	0.994	0.994	0.756
OnLineGames	0.994	0.994	0.219	0.992	0.99	0.006	0.996	0.989	0.438
VBIject	0.993	0.992	0.156	0.995	0.995	0.006	0.995	0.989	0.0
Renos	0.99	0.99	0.0	0.994	0.994	0.013	0.998	0.983	0.0

the classifiers will falsely classify generated malware as actual malware 88 times. Low true negative rate shows that generated malware has such similar characteristics to actual malware that the classifiers are having difficulty in distinguishing between the two. Moreover, the results of the experiments look promising since we applied k-fold cross validation to reduce the effects of overfitting.

## 6 Conclusions and Future Work

In this paper, we aimed at taking advantage of contextualized embeddings created by BERT to generate fake malware embeddings. The generative model we utilized was Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP).

In previous studies, GANs have been shown to generate fake malware opcode sequences. In term of generating malware embeddings, however, there exists a gap in the literature.

We explored that gap in our study by training the generative models on malware embeddings and assess them with three classification models, namely, One-Class SVM (OCSVM), Isolation Forest, and Local Outlier Factor (LoF). The results obtained in our experiments show that WGAN-GP can generate malware embeddings that can closely match the real data distribution. This demonstrates that WGAN-GP algorithms can be successfully applied to produce malware embeddings in addition to generating image data. In some cases, generative models could help increasing the number of data samples for families with limited sample size. This new approach improves the quality of the fake malware generated by GAN algorithms, and creates more encouraging opportunities to apply such data to enhance malware datasets used to train supervised machine learning models.

For future work, this paper may be taken in a variety of different directions. The dataset, for example, may be expanded, and the experiments can include a greater number of malware families. A multi-class generative model can also be explored instead of training distinct WGAN-GP models for each family. Another possible application is to use many different word embeddings techniques to

support WGAN-GP training, such as Word2Vec, ELMo, or different version of BERT. Training BERT model on malware dataset before generating embeddings could also be considered, which may possibly boost the learning of generative models. Finally, because stateful networks can produce intriguing results, experiments using LSTM-GAN can be performed.

## References

1. Advanced guide to inception V3, Google. <https://cloud.google.com/tpu/docs/inception-v3-advanced>
2. Aycock, J.: Computer Viruses and Malware. Springer, New York (2006)
3. Dhanasekar, D., Di Troia, F., Potika, K., Stamp, M.: Detecting encrypted and polymorphic malware using hidden Markov models. In: Parkinson, S., Crampton, A., Hill, R. (eds.) Guide to Vulnerability Analysis for Computer Networks and Systems. CCN, pp. 281–299. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-92624-7\\_12](https://doi.org/10.1007/978-3-319-92624-7_12)
4. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv, abs/1910.01108 (2019)
5. O’Kane, P., Sezer, S., McLaughlin, K.: Obfuscation: the hidden malware. IEEE Secur. Priv. **9**(5), 41–47 (2011). <https://doi.org/10.1109/MSP.2011.98>
6. Hugging Face. Distilbert. [https://huggingface.co/transformers/model\\_doc/distilbert.html](https://huggingface.co/transformers/model_doc/distilbert.html)
7. Clark, K., Khandelwal, U., Levy, O., Manning, C.: What does BERT look at? an analysis of BERT’s attention. In: Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Florence, Italy, August 2019, pp. 276–286. Association for Computational Linguistics (2019)
8. Microsoft Security Intelligence. Winwebsec (2010). <https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fWinwebsec>
9. Microsoft Security Intelligence. Zbot (2010). <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=win32%2Fzbot>
10. Asher-Dotan, L.: What is zero access malware, cybereason i cybersecurity software to end cyber attacks, 16-May-2016. <https://www.cybereason.com/blog/what-is-zeroaccess-malware>
11. Microsoft Security Intelligence. VBInject (2010). <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=VirTool:Win32/VBInject%26ThreatID=-2147367171>
12. Microsoft Security Intelligence. Onlinegames (2008). <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS%3AWin32%2FOnLineGames>
13. Microsoft Security Intelligence. Renos (2006). <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Renos&threatId=16054>
14. Microsoft Security Intelligence. BHO (2020). <https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/BHO.BO>
15. Johnson, J.: Number of malware attacks per year 2020, Statista, 20-Aug-2021. <https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/>
16. Vaswani, A., et al.: Attention is all you need (2017). <https://arxiv.org/abs/1706.03762>

17. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D.: Adversarial machine learning. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 43–58 (2011)
18. Nappa, A., Rafique, M.Z., Caballero, J.: The MALICIA dataset: identification and analysis of drive-by download operations. *Int. J. Inf. Secur.* **14**(1), 15–33 (2015). <https://doi.org/10.1007/s10207-014-0248-7>
19. “novelty and outlier detection”, scikit-learn. [https://scikit-learn.org/stable/modules/outlier\\_detection.html](https://scikit-learn.org/stable/modules/outlier_detection.html)
20. Liu, F.T., Ting, K.M., Zhou, Z.: Isolation forest. Eighth IEEE Int. Conf. Data Min. **2008**, 413–422 (2008). <https://doi.org/10.1109/ICDM.2008.17>
21. Burks, R., Islam, K.A., Lu, Y., Li, J.: Data augmentation with generative models for improved malware detection: a comparative study. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON), pp. 0660–0665 (2019). <https://doi.org/10.1109/UEMCON47517.2019.8993085>
22. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training GANs (2016)
23. Bounsiar, A., Madden, M.G.: One-class support vector machines revisited. In: International Conference on Information Science & Applications (ICISA) 2014, pp. 1–4 (2014). <https://doi.org/10.1109/ICISA.2014.6847442>
24. Pokrajac, D., Lazarevic, A., Latecki, L.J.: Incremental local outlier detection for data streams. In: IEEE Symposium on Computational Intelligence and Data Mining 2007, pp. 504–515 (2007). <https://doi.org/10.1109/CIDM.2007.368917>
25. Kale, A.S., Pandya, V., Di Troia, F., et al.: Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *J. Comput. Virol. Hack. Tech.* (2022). <https://doi.org/10.1007/s11416-022-00424-3>
26. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN (2017)
27. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS 2017, pp. 6629–6640. Curran Associates Inc. (2017)
28. Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: research developments, trends and challenges. *J. Netw. Comput. Appl.* **153**, 102526 (2020). <https://doi.org/10.1016/j.jnca.2019.102526>, <https://www.sciencedirect.com/science/article/pii/S1084804519303868>
29. Lu, Y., Li, J.: Generative adversarial network for improving deep learning based malware classification. In: 2019 Winter Simulation Conference (WSC), pp. 584–593 (2019). <https://doi.org/10.1109/WSC40007.2019.9004932>
30. Roberts, J.M.: VirusShare.com - Because Sharing is Caring (2011). <http://www.virusshare.com>
31. Harshit, T.: Fake malware opcodes generation using HMM and different GAN algorithms (2021). Master’s Projects. 1001. <https://doi.org/10.31979/etd.eq6a-twvq>, <https://scholarworks.sjsu.edu/etd-projects/1001>
32. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.: Improved training of wasserstein GANs (2017)
33. Basole, S., Di Troia, F., Stamp, M.: Multifamily malware models. *J. Comput. Virol. Hacking Tech.* **16**(1), 79–92 (2020). <https://doi.org/10.1007/s11416-019-00345-8>
34. sklearn. Gridsearchcv. [https://scikitlearn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

