Familiarization task

The goal of this task is to find out how the benign and malicious defend data look like and what effect the inner maximizer training method has to create adversarial examples from the malicious data. These adversarial examples are malicious examples that are (slightly) modified with the goal of not being recognized as malicious anymore (attack side).
The data consists of benign and malicious Portable Executables (PE). Each PE is after training represented as a (binary) feature vector where each feature corresponds to a unique Windows API call (22761 in total), indicating its presence (1) or absence (0).

The framework.py file is modified such that it saves the benign, malicious, and adversarial data to separate files. Furthermore, several parameters in the parameters.ini file were changed according to the instructions defined in the assignment. Due to the enormous runtime, it was decided to only include 1.000 out of the 15.200 files for this task. The number of epochs and evasion iterations is set to 50. The attack method is set to the dFGSM.

We first trained the model to analyze the datasets. No model weights are loaded at the start and the model is trained for 50 epochs. We first compare the top 10 features with the biggest differences in both benign-malicious, malicious-adversarial, and benign-adversarial. In Figure 1 we see that the difference between benign-malicious and benign-adversarial is somewhat equal and that the difference between benign-adversarial is much lower. This is of course expected since the adversarial examples should still contain the necessary code that malware needs to successfully execute its purpose.

This is however not a very clear insight into how the datasets actually differ. In Figure 2 the differences in frequencies are displayed for the same situations only for the first 2000 features (the remaining features had a very low frequency with some outliers). It looks like the features are sorted by importance. We see that the previous observation also applies in these plots: malicious-adversarial is very similar and the difference between benign and the other two is well visible. It is important to note that we only take the overall difference into account and not the individual rows (computationally very expensive).

The next part of this task involves training the model for 50 epochs and then train again with the model weights loaded as a starting point. If we look at the adversarial examples compared to the previously generated adversarial examples we see the same trend. We also observe that the difference between malicious and adversarial is even smaller. In the second plot Figure 3, the biggest differences in the top 10 features decreased by more than 2 times. The difference between benign and adversarial however just slightly decreased, meaning the adversarial examples moved towards benign samples. In Figure 4 we see that almost all adversarial points overlap with the malicious points with a slight offset. This means that the effect of extra training and using the model weights of the first training as a base made the adversarial examples move closer to the malicious samples. From this, we can conclude that the model has become more robust to adversarial modification. We can however not say anything about the success rate of the defense/attack.

Figures familiarization task:

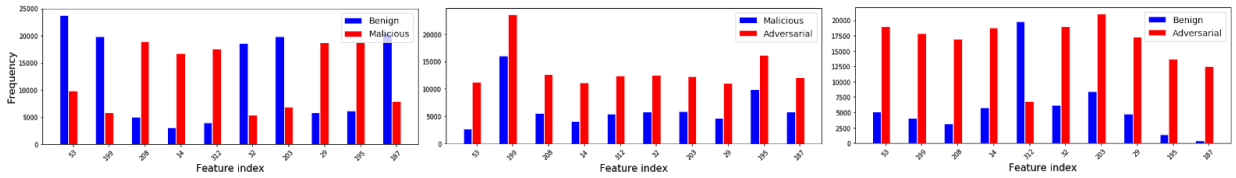

Figure 1: Top 10 features with largest differences in frequencies of noted datasets after simple training

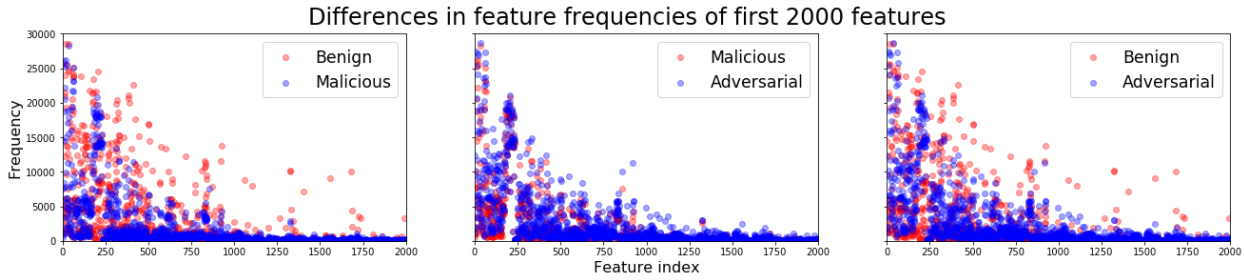

Figure 2: Differences in feature frequencies of first 2000 features of noted datasets after simple training

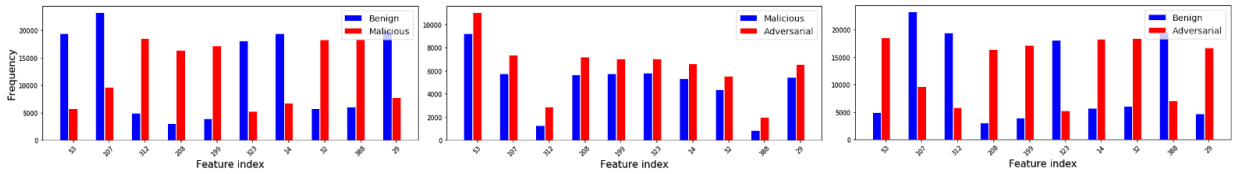

Figure 3: Top 10 features with largest differences in frequencies of noted datasets after intensive training

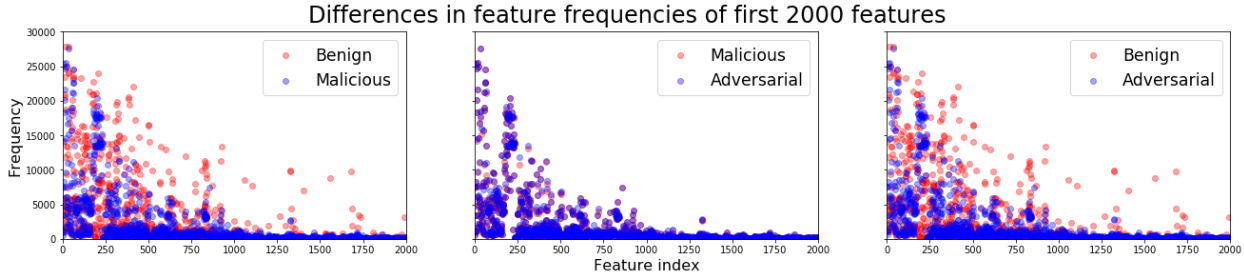

Figure 4: Differences in feature frequencies of first 2000 features of noted datasets after intensive training

Inner maximizer task:
This task involves designing an evasion method that successfully creates adversarial examples out of the malicious samples in which defenses highlight the examples as benign. The inner maximizer file already contains several different evasion methods, such as dFGSM which is used to create those samples in the previous task.

The method, *GRAMS - topk variant* [1], is chosen to create adversarial examples. Since it was difficult to come up with an original method to modify the data (that also includes a good performance), it was decided to implement this method using the pseudocode in the paper to see if we were able to reproduce the increase in performance.

The performance of this model is compared to the performance of the bga_k and rfgsm_k models provided by the teachers, and the dfgsm_k model created in the familiarization task. It is measured by the evasion rate, f1-score, and the run-time of the models. The evasion rate indicates the success rate of the attacker, which is equivalent to the failure rate of the defender. In other words, the ratio of the created adversarial samples that are not marked as malicious by the defending system. For the evasion rates, only the train, test, and validation sets for malicious using the corresponding model are used and equals the percentage of the samples that the defense was not able to recognize as malicious. The f1-score indicates how precise and robust the model is and is composed of precision and recall. For the f1-score the results of the benign samples are also included because otherwise, it isn't possible to compute the f1-score at all (if only malicious samples are included there exists only true positives and false negatives).

In Table 1, the results of the different models can be found. It could easily be seen that the performance of all three metrics is a lot better in the grams model. The evasion rate of the grams model is about three times higher than the other ones. Almost 20% of the adversarial examples are not caught by the defense. The f1-score is likewise a lot higher and results in almost 90%, about 10% higher than the other models. This metric indicates that the model is both quite precise and robust. If we look at the run-time we observe a big decrease in time of at least more than halve. Thus, in all three measures, the grams model is superior.

In the GRAMS paper [1] it was not explained why the default value of the number of bits (with the largest absolute gradient) to flip (k) was chosen to be eight. It is however discussed that the value of k was derived by trial and error. To verify if our model produces the same results we performed two changes in k, by training two grams new models using $k=4$ and $k=16$. The former resulted in a way lower evasion rate of around 0.07, which is just slightly better than the other three models. We can thus conclude that this number of flipped bits is too low to start with. For the latter case, the results are similar. However, the evasion rate is about 0.01 lower and the run-time is about 20 seconds faster. The difference in the f1-score is negligible. We can conclude that from these results indeed a k of 8 is the best choice since the evasion rate should be the highest possible.

Table 1: Evasion rates, F1-scores, and run-times of baseline methods and GRAMS method.

Model	**grams**		**rfgsm_k**		**bga_k**		**dfgsm_k**	
Metrics	**Evasion rate**	**F1-score**	**Evasion rate**	**F1-score**	**Evasion rate**	**F1-score**	**Evasion rate**	**F1-score**
Train set	0.1899	0.9137	0.0702	0.7981	0.0570	0.8060	0.0640	0.7631
Test set	0.1816	0.8884	0.0658	0.8121	0.0553	0.8220	0.0632	0.7748
Val. set	0.1987	0.8795	0.0671	0.8061	0.0539	0.8115	0.0592	0.7570
Overall	0.1901	0.8939	0.0677	0.8055	0.0554	0.8132	0.0621	0.7650
Runtime	185.46 sec		403.96 sec		493.34 sec		403.05 sec	

References:

[1] *The Robust Malware Detection Challenge and Greedy Random Accelerated Multi-Bit Search.* Al-Dujaili et al.

Delivered Files in zip:

- framework.py is modified:
  - Line 105-115, 121-125: for writing the examples to the corresponding files
  - Line 191-194, 234-244, 251-252: for elements of confusion matrix and f1-score
  - Line 470 and 472: for calculation the runtime
- inner_maximizer.py is modified:
  - Line 384-463, 486-487: grams algorithm
- BONUS: [training_grams_evasion_grams]_demo_epoch_49.pt is added as the grams model for bonus task
- BONUS: aes.npy is added as the file with generated adversarial examples