Machine Learning Security Project

Yifei Huang, JiaWei Yao, Hui Cai, Zhaoxiong Chen

Two algorithms, fine-pruning and trojan, are designed for this particular project. First, we show the details of our code, and then analysis the result by comparing those two algorithms.

### Fine-Pruning Detector

The code with fine-pruning defense is based on the idea: clean data and poison data will trigger different kinds of neurons. If we can find the trigger pattern of clean data or poison data, we can use it to classify an unseen data. To achieve this goal, fine-pruning algorithm is applied on the clean validation data. Pruning algorithm tells us that if we prune neurons in ascent order of activation value, the neurons that are activated by neither clean nor backdoored inputs are pruned firstly, then comes to that are activated by the backdoor but not by clean inputs and finally is the that are activated by clean inputs. Based on this observation, we can divide the whole neurons into three parts: invalid neurons, backdoor neurons and clean neurons. Therefore, we can extract the clean neurons, calculate some metric on these neurons and use it for clean pattern. For a new data, we can measure whether it follows the pattern. Now, we can come up with the idea of our defense attempt:

1. For a bad-net input and the clean validation data, using pruning algorithm to find the *fraction of neurons pruned – accuracy* figure. Analysis the figure, and the number of invalid neurons is calculated by the fraction corresponding to 4% drop on accuracy.
2. Set the number of clean neurons as a hyperparameter. The rest is the backdoor neurons. By sorting the activation value of validation data activation\_val\_*sort*, the pick the highest neuron index *clean\_mask*.
3. Calculate the average activation of the clean neurons on clean validation data, *activation\_val[clean\_mask]*.
4. For an unseen data, first feed it into the bad-net and get the activation on clean neurons, *x\_test[clean\_mask]*. If the difference between *x\_test[clean\_mask]* and *activation\_val[clean\_mask]* is larger than a threshold which is also a hyperparameter, it means this data is much different from the clean validation data, which is labeled as the poisoned N+1 class (with label = -1).
5. Using validation data to fine-tunning the pruned model. And using this model to predict the data which is not labeled as poison in step 4.

For this particular project, the test set is only given for bad-net 1. Therefore, we only use it for choosing hyperparameters. After analysis, 20 and 0.8 is selected as number of clean neurons and threshold, respectively. For bad-net 2 and bad-net 3, we just use the same hyperparameters.

Pros and cons: On the one hand, there are no clear edges between invalid neurons, backdoor neurons and clean neurons. It is likely that the pattern of clean neurons are similar to the backdoor one, in which case we cannot effectively select the poison data. On the other hand, if we don’t know the exact feature of the backdoor attack, it will be impossible for us to selecting hyperparameters. Using hyperparameters of another bad-nets may result in a low detect accuracy.

## STRIP Detector

Apart from fine-pruning, we tried another approach called STRIP (STRong Intentional Perturbation) to detect unknown backdoor and separate backdoored inputs in a recent study[1]. The basic idea of the method is perturbing the incoming inputs intentionally by superimposing various image patterns, then observing the randomness of the predicted classes. In terms of malicious inputs, regardless of strong perturbations, the predictions of them tend to be always consistent and fall into the preset class. We will explain these steps in our experiments in detail.

In the code, we processed BadNet1 and its inputs first, then we used similar steps to repair BadNet2 and BadNet3.

### Repairing Sunglasses Poisoned BadNet (B1)

For a BadNet, firstly, we need to perturb the inputs. We replicated every given incoming input N times and got N perturbed inputs by superimposing the image of both the input copy and an image randomly extracting from clean inputs.

Secondly, these perturbed inputs were fed into the BadNet and it would output N predicted results. Then we calculated the entropy to evaluate the randomness of the predicted classes of the perturbed inputs of the above original input. For the BadNet1, since we knew the malicious inputs and clean inputs, we can observe the distribution of the entropy of these two kinds of inputs. We tuned the value of N (the number of perturbed replica of an incoming input) and observe the impact on entropy distribution of clean and backdoored inputs in the following figure.

|  |  |
| --- | --- |
| Figure 1. Entropy distribution of clean and malicious inputs | Figure 2. Value of FRR and FAR |

We chose 100 as the number of replicas. Since the perturbation with 100 images could achieve nearly the same result generated by 1000 perturbed images, which was good enough. Also, perturbation with 100 images had a much smaller computation overhead.

From the entropy distributions, we found given a BadNet, the backdoor tended to be triggered when the model predicted perturbed malicious input. The BadNet behaved just like what attackers expected, which generated concentrated results that had lower entropy, as we could see in the orange part. While perturbed samples of clean inputs would not trigger the backdoor and their prediction results followed a normal distribution, and the entropy values were higher than that of malicious inputs.

According to the above analysis, the next step is to decide the threshold of entropy value to separate the malicious inputs and clean inputs. Since the entropy values ​​of the two parts have a little bit of overlap and we don’t have the malicious inputs when we process the BadNet2, we cannot use the lowest value of entropy of clean inputs as the threshold. Instead, we can determine FRR at first, then calculate the percentile of the normal distribution based on the results of clean inputs. In fact, we used the percentile as the boundary to detect backdoored inputs. For the BadNet1, we tried different values of FRR to improve this trade-off between FRR and FAR. The results are represented as the following figure. Due to the space limitation of this report, table about FRR, FAR and threshold are provided in the GitHub repository.

According to the experiment data, we chose **0.2942637391590416** as the threshold.

After that, we could repair the BadNet1 and call the repaired version G1. An incoming input with the entropy results lower than the threshold would be judged as the backdoored input and the model would output class -1.

To measure the performance of G1, we used 3 validation sets:

1. 1000 samples from clean validation set and 1000 samples from poisoned data set. Label ground truth of all poisoned samples are set as -1;
2. Entire clean validation set;
3. Entire sunglasses poisoned set.

For validation set 1, we got an accuracy of 94.75%. For validation set 2, we got an accuracy of 91.64%. For validation set 3, we got an accuracy of 3.6%. The result matches what we expected: a low prediction accuracy on backdoored samples, while a high prediction accuracy on clean samples. Therefore, we can see the method STRIP works well on BadNet1 and we get a good “repaired” net.

### Repairing Anonymous Poisoned BadNet (B2)

In terms of the BadNet2, we processed the inputs in the validation datasets as the previous perturbation steps and draw the entropy distribution of the clean inputs in the colab.

Since the entropy distribution is similar to the distribution in Figure 1, and given that there is only one backdoor target output, we could expect that the entropy distribution of perturbed poisoned samples could be similar to that of sunglasses poisoned samples. This means that we may keep using the same threshold value as we obtained in the BadNet1. Still, we need to build perturbation on an incoming input when detecting backdoors, feed the input to the repaired net, calculate the entropy summation of the output, and compare the result with our detection threshold. If the result is lower than the threshold, the net will output class -1.

### Repairing Eyebrow Poisoned BadNet (B3)

For fixing eyebrow poisoned BadNet B3, what we have done is nearly the same as that in repairing Sunglasses Poisoned BadNet B1, since we had the poisoned example along with clean example. Therefore, in this section we will focus more on the result.

Firstly, there is some significant change in normalized entropy distribution. It could be obtained that due to multiple backdoor targets, the entropy of perturbed poisoned samples has increased, which make it harder to find a proper threshold without increasing too much on FRR and FAR.

Secondly, the FRR and FAR values are also raised. The following figure presents the FAR curve with the increase of FRR value. Through selecting the optimal threshold, the smallest FRR and FAR value we could achieve is nearly 20% each. The table of FRR, FAR and threshold is also provided in the GitHub repository.

Finally, we repaired the BadNet B3 using the selected threshold and measured the performance of repaired network G3 with 2 datasets: the clean validation set and the whole eyebrow poisoned set and. For the first dataset, we got an accuracy of 80.55%, and for the second dataset, we got an accuracy of 9.90%. These data shows that more benign samples are classified incorrectly while more trojaned samples are classified as benign samples.

### Conclusion

STRIP method for backdoored sample detection achieved satisfying performance in backdoor attacks with single target. For situation like backdoor attack with multiple triggers and multiple targets, the performance has dropped quite a lot. The possible reason is that the way STRIP distinguishes backdoored sample is based on entropy, and the entropy reveals the uncertainty. Since multiple target backdoor outputs have more uncertainty, it makes the entropy distribution of perturbed poisoned samples closer than that of perturbed clean samples.

### Reference

[1] Gao, Yansong, et al. "Strip: A defence against trojan attacks on deep neural networks." Proceedings of the 35th Annual Computer Security Applications Conference. 2019.