Finding Tempered Training Data by using Gradient Tracing

Ziyad Yacoubou

28 April 2024

Abstract

In machine learning, influence is defined as, given training data, in what way does the training data influence the model and its prediction. There are multiple ways to compute the influence, but in this paper the method of influence computation will be done using a method called TraceIn [2]. TraceIn method essentially computes the influence using the model and gradients. Using the calculated influence, you can eliminate or detect data which has been tempered with. Tempered data should have a high influence value, as opposed to non-tempered data. The goal is to demonstrate to paper is to demonstrate whether TraceIn is capable of identifying data which has been tempered with.

1 Introduction

In 2023 Nvidia's stocks increased by significant amount due to the large boom in A.I. Over the past few months the usage of A.I. has also increased. Many people are using generative pre-trained transformers (GPT) such as ChatGPT and Microsoft Copilot for work and personal reasons. A.I. can be used to generate art or other purposes. Overall, A.I. has seen a large amount of usage increase. With increase in usage, came with it multiple security and usage problems and topics of discussion. A.I. models are based entirely on the data that is used to train it. This is a limitation and also an Achilles heel. If you train a model with false or malicious information, then the user using that model will get false predictions and solutions. Some data can also be extracted from models. These security issues are important. it is imperative that A.I. models must be secure and robust. This is due to the fact some fields have high usage rate, so incorrect predictions are dangerous, and some models are trained using private information.

There are a multitude of security attacks that can be implemented against A.I. machine learning models, but for this paper the main focus will be a white-box attack. A white-box attack is when the attacker has access to the model and training data. A type of white-box attack is data poisoning. Data poisoning is

when the attacker poisons the training data, so that when the model is trained, it makes the wrong predictions. The poison which this paper covers is label poisoning, specifcally label flipping and clean label poisoning [1].

2 Poison Attack

2.1 Label Flipping

Label is flipping is one way to implement label poisoning. Label flipping is targeted attack. It is really simple in nature. All you do is find a target and flip its label to another label which is in the dataset.For this experiment, two variants of label flipping were implemented.Targeted label flipping 3 and 9 of the MNIST dataset were used because they look somewhat similar to each other. The other implementation was selecting random labels and flipping them to a random label.

2.2 Clean-Label Poisoning

Clean label poisoning is the opposite of label flipping. The training is manipulated to make the wrong predictions without changing the labels. Small perturbations are added to the image, such that when the model predicts using the image, it is incorrect. One method of crafting clean label poisoned data is to use feature collision. This paper [1] shows how feature collision and poison crafting is accomplished. According to the paper, in Feature Collision, you want to find a $\mathbf{x}$ that collides with the target in feature space but is close to the base instance $\mathbf{b}$. This is given by the equation

$$\mathbf{p} = \underset{x}{argmin}\ ||f(\mathbf{x}) - f(\mathbf{t})||_2^2 + \beta||\mathbf{x} - \mathbf{b}||_2^2$$

In order to perform this attack in the experiment the Adversarial Robustness Toolbox (ART) was used to perform the feature collision [3].

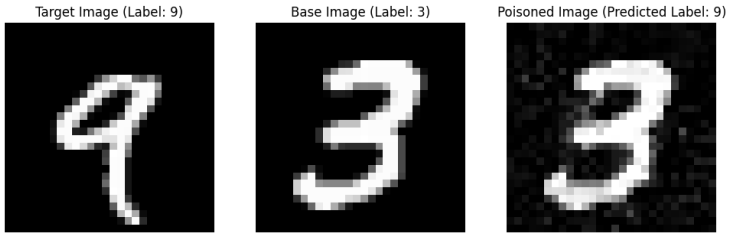

Figure 1: Feature Collision

3 TraceIn Method

3.1 What is the TraceIn Method?

The TraceIn method is a way to compute the influence of a training point. The trace method is based upon the fundamental theorem of calculus. There are multiple ways to compute influence. One method is the representor point method and influence function [2]. These solutions are impractical for this experiment because they rely on an optimality condition. The influence function requires that you compute the Hessian which can be memory intensive. So then, why TraceIN? TraceIn only relies on gradient, checkpoints, and loss function of the model. With TraceIn, you can use first order approximation, saved checkpoints, and you can cherry-pick specific layers. This makes TraceIn really scalable. TraceIn works with any model that has been trained using stochastic gradient descent.

3.2 How is TraceIn Computed

Recall that TraceIn is based on the fundamental theorem of calculus. The Ideal trace is defined as

$$TraceInIdeal(z, z') = \sum_{t:z_t=z} \ell(w_t, z') - \ell(w_{t+1}, z')$$

This formula is saying that given training point z, test point z', weights w, and loss function ℓ, the TracInIdeal is the sum of the loss of the current iteration subtracted by the next iteration in the training process. This formula is inefficient. One reason for each training point, you would have to keep track of the loss of the current and next iterations in the training.This will increase the time of execution because you have to replay the training. Another reason is that this formula only take into account one training point and one testing, and most data are loaded in batches. To address the batches, the paper [2] use this formula.

$$TraceIn(z, z') = \frac{1}{b} \sum_{t:z\in B_t} \eta_t \nabla\ell(w_t, z') \cdot \nabla\ell(w_t, z)$$

Notice the slight change in the formula. Now instead of subtracting the losses, first order approximation is used. In the summation, the product is multiplied by, η which is the learning rate. After computing the sum, you divide by the batch size b. This solves the batch problem, but still does not address the time consumption. In the order to address the time consumption, the paper introduces TraceInCP.

$$TraceInCP(z, z') = \sum_{i=1}^{k} \eta_t \nabla\ell(w_t, z') \cdot \nabla\ell(w_t, z)$$

This is similar to TraceIn, but you iterate over k checkpoints. This handles the time consumption because you are not replaying the training. You can

implement this by saving checkpoints as you train the model. This flexible because it allows to pick when to save the checkpoints. TraceIP is the version that is used for this experiment.

4 Evaluation

For the experiment, two resnet18 models were trained with the MNIST dataset. One of the resnet18 model was clean and the other was trained using poisoned data via clean label poisoning. Both Models were trained with 50 epochs and checkpoints were saved every 5 epochs. Using 100 Training points which have been poisoned and 20 test points, we compute the TraceInCP value. To decide whether the data has to be tempered, a threshold γ was used. For this experiment $\gamma = 300$

Algorithm 1 Tempered Data Algorithm

```
for i = 1 to tr do                                   ▷ train images
    CP ← 0
    for j = 1 to te do                               ▷ test images
        CP = CP + TraceInCP(i,j)
    end for
    if CP ≥ γ then
        Tempered ← True
    end if
end for
```

4.1 Data

The tables below are the results of the experiment. The table shows, of the small portion of 100 training points that have been poisoned, how many were detected by the TraceInCP.

Poison Type	Percentage of Poison Data Detected
Targeted Label Flipping	100%
Random Label Flipping	85%
Clean Label Poisoning	20%

Table 1: Clean Resnet18

Poison Type	Percentage of Poison Data Detected
Targeted Label Flipping	100%
Random Label Flipping	80%
Clean Label Poisoning	6.25%

Table 2: Poisoned Resnet18

5 Discussion

From the data it can be concluded when it comes to simple label flipping, TraceInCP is able to detect mislabeled data. Where there is a significant amount of failure is when clean label poisoning is performed. Also, notice the poisoned model performs slightly worse than the clean model. From these observations, it can be concluded that TraceInCP works best when data is mislabeled. It will not work for clean label poisoning. Another takeaway is that for TraceInCP to work, the model has to be a clean model. If the model is poisoned, then the TraceInCP does not work as intended.

6 Conclusion

In this paper, we test TraceInCP to see if it is capable of detecting data which have been tempered with. From the experiment it can be seen that TraceIn method works for simple label flipping but fails, when clean label poisoning. Detecting Clean label poisoned data cannot be done by TraceInCP. We did not test other influence calculation methods, but TraceInCP does not work. Ideally TraceInCP can only be used to find mislabeled data, and that is its limitation.

References

[1] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, Tom Goldstein. (2018).*Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks*. https://arxiv.org/abs/1804.00792

[2] Garima Pruthi, Frederick Liu, Mukund Sundararajan, Satyen Kale . (2020). *Estimating Training Data Influence by Tracing Gradient Descent*. https://arxiv.org/abs/2002.08484

[3] https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/attacks.html

[4] Source Code. https://github.com/Ziyad-Y/TraceIn-Posion-Detection