Preventing Clean Label Poisoning using Gaussian Mixture Loss

Muhammad Yaseen* Muneeb Aadil* Maria Sargsyan*
Max Planck Institute for Informatics, Saarbrücken, Germany
{myaseen, maadil, msargsyan}@mpi-inf.mpg.de

Abstract

Since 2014 when Szegedy et al. [6] showed that carefully designed perturbations of the input can lead Deep Neural Networks (DNNs) to wrongly classify its label, there has been an ongoing research to make DNNs more robust to such malicious perturbations. In this work, we consider a poisoning attack called Clean Labeling poisoning attack (CLPA) [4]. The goal of CLPA is to inject seemingly benign instances which can drastically change decision boundary of the DNN due to which subsequent queries at test time can be mis-classified. We argue that a strong defense against CLPA can be embedded into the model during the training by imposing features of the network to follow a Large Margin Gaussian Mixture distribution in the penultimate layer. By having such a prior knowledge, we can systematically evaluate how unusual the example is, given the label it is claiming to be. We demonstrate our builtin defense via experiments on MNIST and CIFAR datasets. We train two models on each dataset: one trained via softmax, another via LGM [7]. We show that using LGM [7] can substantially reduce the effectiveness of CLPA while having no additional overhead of data sanitization. The code to reproduce our results is available online.

1. Introduction

With the ubiquity of Deep Neural Networks (DNNs), the issues concerning their security are becoming more and more relevant. There is thus an increasing interest in the research community to create Neural Networks which achieve state-of-the-art results, while also being secure, private, and robust. It is well known that DNNs are vulnerable to adversarial perturbations [6] and an adversary might corrupt the input imperceptibly but maliciously which would change the classification result.

One such class of adversarial attacks is clean-label poisoning attacks (CLPA). In CLPA, an attacker constructs a poison training instances which looks like one class to human, but like another class to DNN. The aftermath of poisoning is that the attacker can then, during test time, can query the DNN with the malignant class which could be mis-classified as benign. Thus, effectively, a malignant instance can surpass the security mechanism. We try to tackle such an attack exploiting the intuition that specially constructed poisoning instances are far away from the class distribution in the feature space of the class they're claiming to be. That is why, we impose Gaussian Mixture distribution on the features. Experimentally, we show that it is relatively difficult to generate adversarial examples for our model. To our knowledge, this is the first model which embeds the CLPA defense into the network itself without requiring additional overhead of data sanitization.

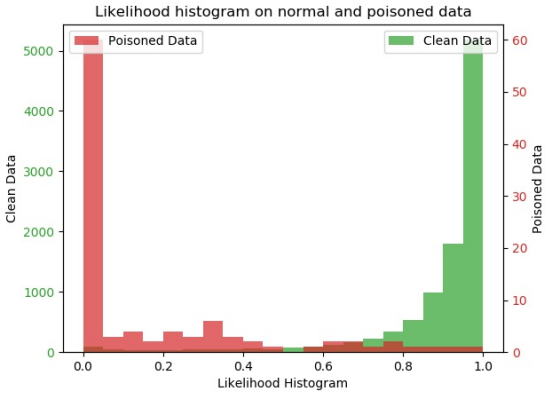


Figure 1: Distribution of likelihood values for cleaned (MNIST Test Set) and poisoned data: Poisoned and cleaned instances have mostly low and high likelihood, respectively. This suggests we can differentiate confidently b/w cleaned and poisoned instances via likelihood thresholding.

2. Background

In this section, we explain the background relevant to our model. First, we formally define CLPA [4]. Secondly, we review LGM [7] and explain how likelihood of the features

*denotes equal contribution.

can be computed.

2.1. Clean-Label Poisoning attacks

In [4] authors introduced the notion of *Clean Labelling Poisoning Attack* (CLPA) which we describe below.

Let Alice be an adversary, Bob be a potential victim. Suppose that Charlie trains a huge network $F(x)$ on a gigantic cloud dataset D_C and uploads the weights online. Further assume that Bob wishes to finetune the model $F(x)$ for some task for which the clean finetuning dataset $D_f = (X_i, Y_i)_{i=1}^{n_f}$ is available online. However, Alice constructs a poisoned dataset $D_p = (X_i, Y_i)_{i=1}^{n_p}$ and uploads it amid the D_f to construct total finetuning dataset $D_t = D_f \cup D_p$. Notice that Bob is unaware of the poisoned instances, since he will download the available data online (which also potentially includes D_p). Next, Bob will train his model $f'(x)$ on D_t which can potentially alter the otherwise reasonable decision boundary into vulnerable one, allowing subsequent target (usually malignant) class to be misclassified into base (usually benign) class (see figure 7 in appendix A).

In [4], the authors show how easy it is to make a clean-label targeted attack to the class of models trained by transfer learning techniques just with a single crafted examples. More specifically, to create the poisoned examples, authors optimize the following objective:

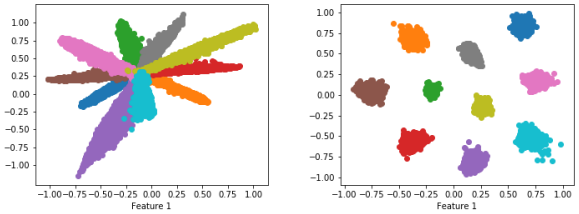
$$\mathbf{p} = \arg\min_x \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta\|\mathbf{x} - \mathbf{b}\|_2^2 \qquad (1)$$

Where t and b are target and base image respectively. $f(x)$ represents the activations of penultimate layer, and β is a trade-off parameter. As such, the *image* of constructed poison is similar to base instance, while its *features* resemble that of target instance.

2.2. Gaussian Mixture Loss

In [7] authors introduced a new approach to make feature distributions more structured by incorporating some prior assumptions about features in the loss function. They assume the features of penultimate layer to be realizations from a mixture of K Gaussians corresponding to K classes. During training, this Gaussian structure is enforced by incorporating a loss term which measures the deviation from distribution and penalizes proportionally. The model learns the means of K Gaussians as parameters which is encouraged to have large inter-distribution distance (α). For simplicity, we used only isotropic mixture of Gaussians in this project i.e. the co-variance matrix is set to identity.

The modelling assumptions are as follows: Features $\boldsymbol{x}$ of a particular class are assumed to have the density shown in Eq 2. The class conditional density i.e. distribution of features give the label is then given by Eq 3. The posterior probability of class given feature is thus obtained via Bayes rule as shown in Eq 4. This enables us to get the likelihood of an example belonging to a class given its features.



(a) Softmax Loss (b) LGM Loss ($\lambda = 1$)

Figure 2: Feature Distribution of MNIST Training Set: Features are far apart for LGM Loss than for Softmax Loss. Different classes are color coded.

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \mathcal{N}(x; \mu_k, \Sigma_k)p(k) \qquad (2)$$

$$p(\boldsymbol{x_i}|\boldsymbol{z_i}) = \mathcal{N}(x_i; \mu_{z_i}, \Sigma_{z_i}) \qquad (3)$$

$$p(\boldsymbol{z_i}|\boldsymbol{x_i}) = \frac{\mathcal{N}(x_i; \mu_{z_i}, \Sigma_{z_i})p(z_i)}{\sum_{k=1}^{K} \mathcal{N}(x; \mu_k, \Sigma_k)p(k)} \qquad (4)$$

Under the assumptions described above, the large-margin Gaussian mixture loss is given in Eq 5. It consists of two components: (1) $\mathcal{L}_{cls}$ (softmax loss) and (2) $\mathcal{L}_{lkd}$ (deviation from Gaussian distribution) and λ is a trade-off parameter.

$$\mathcal{L}_{GM} = \mathcal{L}_{cls} + \lambda\mathcal{L}_{lkd} \qquad (5)$$

For further details, we refer the reader to [7].

3. Proposed Method

As explained in section 2.1, the attacker generates a poison instance (x_p, y_p) such that the features $f(x_p)$ of different classes in a softmax pre-trained model get close-by. However, since there is no way to query the likelihood $p(f(x_p)|y_p)$, we cannot systematically know how "unusual" the example x_p is for the class y_p that the poisoned instance is claiming to be.

To this end, we use LGM loss [7] to get likelihood $p(f(x_p)|y_p)$ of an example belonging to the class y_p it is claiming to be. The intuition is that poisoned instances (x_p, y_p) are far away from their claimed class in the feature space. Thus, poisoned examples will have low likelihood using which we can remove such suspicious instances before fine-tuning the model. The complete proposed procedure is highlighted in the threat model in algorithm 1.

Figure 3: Comparison of Constructed Poisons on MNIST Test Set for Softmax and LGM. Each set of 4 pictures contains (from left to right): b (base), t (target), p_{CE} (poison for softmax), p_{LGM} (poison for LGM). For each set, notice that P_{CE} is imperceptible to human, unlike P_{LGM} which has noticeable artifacts. This suggests that for $F_{LGM}(x)$, constructing imperceptible poisons is relatively more challenging.

Algorithm 1: Our Contributions (Bold) in Attack Model

1. $F_{LGM}(x)$ **is pre-trained on** D_C **via LGM loss**
2. Alice generates poisoned dataset D_p
3. D_p is mixed along D_f to create $D_t = D_f \cup D_p$
4. Bob downloads D_t and $F_{LGM}(x)$
5. **Bob constructs filtered clean dataset** $D_w = (X_i, Y_i)_{i=1} \forall i, \ldots, n_t \; s.t.$ $p(F_{LGM}(X_i)|Y_i) > T$
6. Bob can now fine-tune his model $f'(x)$ on D_w

4. Experiments and Results

In this section, we describe our experimental details and present the results of our proposed methodology. The code to reproduce the following experiments is available online[1].

4.1. Datasets and Simulation Strategy

We use two standard datasets: MNIST [2], and CIFAR10 [1]. While the original poisoning paper [5] used ImageNet [3] dataset, we skipped it because of computational constraints.

Please note that in the following experiments, we treat training sets as D_C, while test sets as D_f. As such, the proposed methodology corresponds to training base networks (which can be thought of as pretrained networks $F(x)$) on D_C, while creating poisons D_p on test sets (since in real life, attackers poison on the D_f).

[1] https://github.com/muneebaadil/likelihoods-for-poison

4.2. Training Base Models

To check if training a model with LGM loss [7] serves as a good prevention mechanism against clean labelling poisoning attacks, we train two identical CNNs; one with standard cross entropy loss ($F_{CE}(x)$) and another with LGM ($F_{LGM}(x)$). The intuition of doing so is such that if our hypothesis is correct, we should see a difficulty in generating poisons for $F_{LGM}(x)$ as opposed to $F_{CE}(x)$. These two models now can be thought of as pre-trained models $F(x)$ trained on cloud dataset D_C. Due to brevity, we do not describe the network architecture here; however, it is presented in Appendix C for the interested readers.

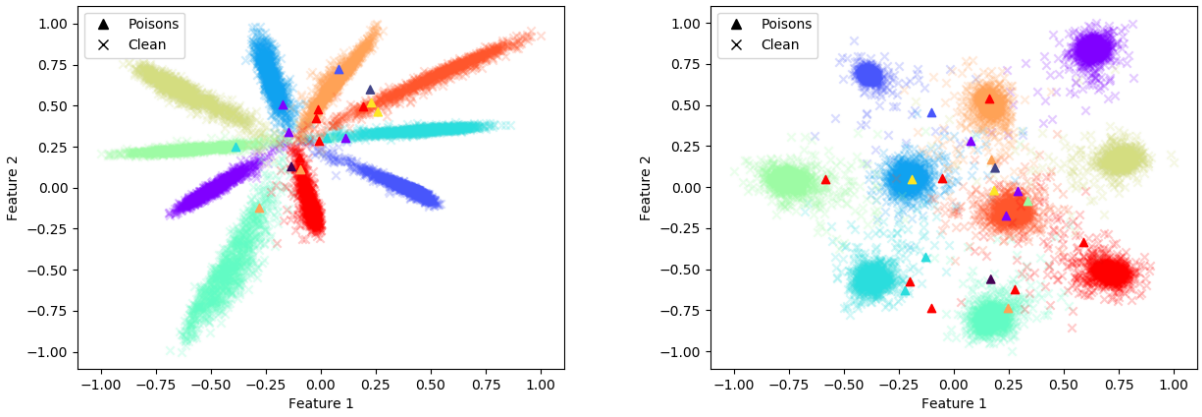
We train both networks until convergence; the feature distribution for both loss functions is shown in figure 2. Notice that features are far apart for different classes in LGM loss unlike standard cross entropy loss. Thus, loosely speaking, it should be relatively difficult than softmax to change the features of base class to resemble target class *while maintaining similarity to base class in the image space*.

4.3. Generating Poisons

Once the base models $F_{CE}(x)$ and $F_{LGM}(x)$ are trained, we implemented poisoning algorithm according to [4] to construct poisons for both base models to evaluate if and how much is there a difference in poisoning examples, when $F_{LGM}(x)$ is employed.

As the threat model assumes that the adversary can only inject 10% of the data into D_f, we constructed 100 poisoning instances D_p to inject inside D_f (which is, in our case, test-sets of MNIST/CIFAR10)[2]. For each poisoning instance, target image t and base image b was chosen randomly. And similarly as in the original paper [4], we

[2] Although, 10% of our test-sets is 1000, we only constructed 100 because of computational constraints.



(a) Softmax Loss

(b) LGM Loss ($\lambda = 0.1$)

Figure 4: Feature Distribution of MNIST Test Set and Poisoned Examples: Clean and poisoned instances are color coded by their ground truth class and base class respectively. Note that only 20 random poisons are shown for clarity.

set $maxIters = 1000$ to construct a poison instance. Lastly, we cross-validated β parameter in the algorithm and empirically found $\beta = 8\text{e}{-3}$ to be the best performing one.

Figure 3 compares visual examples of constructed poisons for $F_{LGM}(x)$ and $F_{CE}(x)$ on MNIST.[3]. Notice that for each set of base b and target t, p_{CE} is much less noticeable of an adversarial example than p_{LGM}, thereby suggesting F_{LGM} to be more robust against poisoning. We argue this is because $F_{LGM}(x)$ features are far apart for different classes, which makes changing a feature representation *without significant changes to image* challenging. Furthermore, figure 4 shows the constructed poisoned instances in feature space.

4.4. Using Likelihood to Filter out Poisons

One direct benefit of using a structured feature representation such as Gaussian is that we can evaluate the posterior likelihood of features given the label $p(f(x)|y)$. Thus, we can ask the model that under learned representation, how likely this feature will be encountered in a class. This property of LGM can be leveraged to prevent CLPA by simply using the likelihood as an inherent trust score of an (image,label) pair and we can discard any impostor example where $p(F_{LGM}(x)|y_{claimed}) < T$.

Figure 1 shows that poisoned and normal inputs are well separated in the likelihood space and thus can be easily distinguished. This is further confirmed by the ROC plot in Figure 5 plotted over different threshold levels.

[3]Due to limited space, we put CIFAR10 results on Appendix B

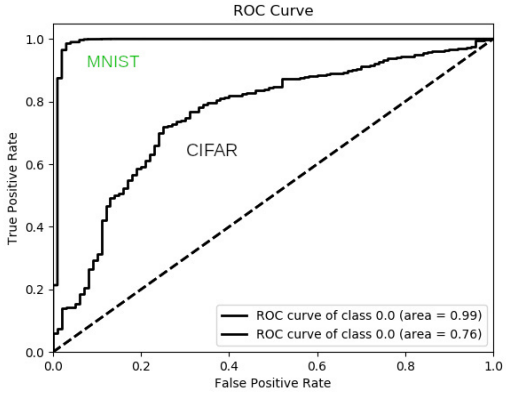


Figure 5: ROC Curve for different likelihood thresholds to filter out poisons.

5. Conclusion

In this work, we showed that structured feature distributions such as mixture of Gaussians substantially restrict the effectiveness of CLPA by making it more challenging to construct clean poisons. It also additionally provides the ability to query feature likelihood which again helps in filtering the potentially poisonous examples. We demonstrate our techniques on two datasets i.e. MNIST and CIFAR-10. The constructed poisons on both datasets under LGM loss are visibly very perturbed and would fail to pass as clean labels. Additionally, we also show that even if the poison is created, network is successfully able to detect it by thresholding the feature likelihood thus preventing the attack.

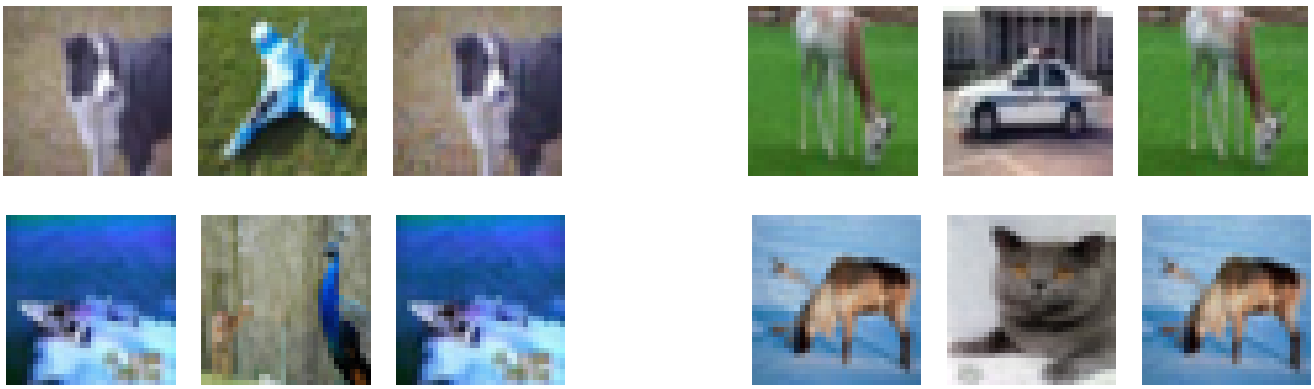


Figure 6: Constructed Poisons on CIFAR Test Set for LGM. Each set of 3 pictures contains (from left to right): b (base), t (target), p_{LGM} (poison for LGM).

References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[2] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[4] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.*, 2018.

[5] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*, 2018.

[6] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[7] Weitao Wan, Yuanyi Zhong, Tianpeng Li, and Jiansheng Chen. Rethinking feature distribution for loss functions in image classification. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9117–9126, 2018.

A. CLPA Demonstration

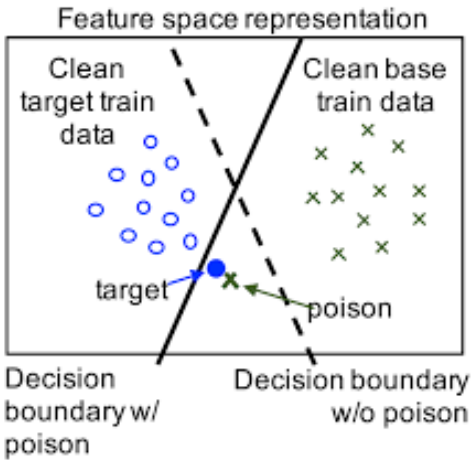


Figure 7: Demonstration of CLPA: A crafted poison instance changes the otherwise decent decision boundary, which mis-classifies a target into base. Figure taken from [4]

B. Constructed Poisons for CIFAR10

Figure 6 shows constructed poisons for CIFAR-10 dataset for $F_{LGM}(x)$; figure 8 displays the likelihood statistics of clean and poisoned data.

C. Neural Network Architecture

C.1. MNIST

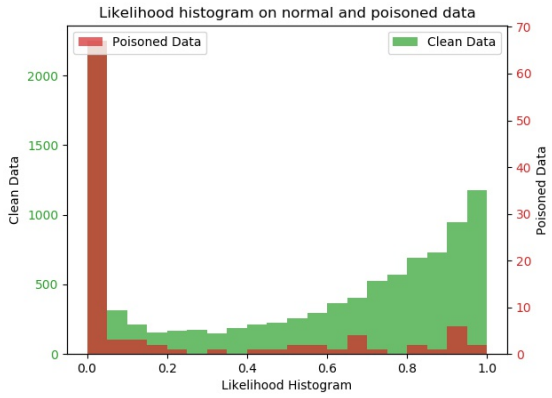


Figure 8: Likelihood histogram for CIFAR10. Our CIFAR10 model wasn't convergent and we couldn't explore it further because of time limit.

MNIST Architecture for LGM	**Output Shape**	**Param #**
Conv2d-1	[-1, 32, 28, 28]	832
PReLU-2	[-1, 32, 28, 28]	1
Conv2d-3	[-1, 32, 28, 28]	25,632
PReLU-4	[-1, 32, 28, 28]	1
MaxPool2d-5	[-1, 32, 14, 14]	0
Conv2d-6	[-1, 64, 14, 14]	51,264
PReLU-7	[-1, 64, 14, 14]	1
Conv2d-8	[-1, 64, 14, 14]	102,464
PReLU-9	[-1, 64, 14, 14]	1
MaxPool2d-10	[-1, 64, 7, 7]	0
Conv2d-11	[-1, 128, 7, 7]	204,928
PReLU-12	[-1, 128, 7, 7]	1
Conv2d-13	[-1, 128, 7, 7]	409,728
PReLU-14	[-1, 128, 7, 7]	1
MaxPool2d-15	[-1, 128, 3, 3]	0
Flatten-16	[-1, 1152]	0
PReLU-17	[-1, 1152]	1
Linear-18	[-1, 2]	2,306
Linear-19	[-1, 10]	30
LGM	[-1, 10]	0
SoftMax	[-1, 10]	0

C.2. CIFAR10

We used VGG19 Architecture for training CIFAR10.