Towards Class-Shielding Data Poisoning

Thomas Pinkava
pinkavat@uw.edu
University of Washington
Bothell, Washington, USA

ABSTRACT

Data Poisoning is the art of altering data to affect the performance of models trained thereon. The domain comprises an arms race between ever-more-devious poisoning techniques ([14],[10], [6], [9],[11]) and ever-more-sophisticated defences ([17], [4][2], [1], [7], [11]) across myriad subdomains and data modalities of machine learning; hence, it is not profitable to devise yet another poison or defence.

Our work focuses on another approach: the devising of a Class-Shielding strategy. In this strategy, a victim model is poisoned such that it misclassifies a 'protected class'. However, the poisoned data are not members of that class – and are engineered to be correctly classified as the classes they purport to be.

We implemented a metalearner-based Deep Neural Network designed to devise poisons for any protected class, given a black-box target model. Our results were unfavourable.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

neural networks, data poisoning, metalearning, image recognition

1 INTRODUCTION

"Data Poisoning", according to a survey by Cinà et al.[3], is the modification of training data in order to compromise the performance of a machine learning model: a model's integrity or availability may be reduced, the model may be coaxed into revealing private training data, watermarks may be inserted into the model's output, or trojans may be injected to change the model's output given specific inputs.

Developing methods for poison generation is a broad and active field. Techniques range from the general, such as Jagielski et al.'s subpopulation attacks (in which similar data are corrupted together to evade detection)[10], to the highly specific, such as Shafahi et al.'s PoisonFrogs, in which tainted data items are engineered to be close to existing data items in featurespace [14].

Poisons are devised for both black-box (such as Huang et al.'s metalearning-based MetaPoison [9]) and white-box models (Geiping et al.'s Witches' Brew is based on gradient matching, in which the poisoned data synchronizes with the victim model's optimization process, to devastating effect [6]).

Means for defence against poison are as many and varied as the poisons themselves: for instance, Aladag et al. use Autoencoders to preprocess model input [1], Borgnia et al. use traditional data augmentation techniques [2], and Geiping et al. use adversarial training to strengthen models against potential poison [7]. Verifying data quality is also an explored domain, such as Fang et al.'s application of 'truth discovery' techniques to crowdsourced data [4] and Zhao et al.'s use of online data-source quality learning as a guard against poison [17]. Specialized defences against specific poisioning tasks also exist, such as Gao et al.'s STRIP, which defends against the implantation of trojans [5].

The arms-race nature of data-poisoning research means that a 'better poison' or 'better antidote' is of evanescent value. More promising is research into *strategies*, that is, into the context surrounding the use of data poison. Sun et al., in their paper introducing the CoProtector repository poisoner [16], show a poisoned code repository divided into 'protected' (true code with poison), 'intensive poison' (nothing but posion), and 'bluff' (unpoisoned code labeled as if poisoned); this arrangement is designed as if to anticipate a sophisticated scraper discarding repositories with detected poison, or learning to identify the poison provided by CoProtector.

The goal of this paper is to devise such a strategy. We aim to create a *Class-Shielding Data Poisoner*, that is, a deep-learning model that, given a class of data one wishes to 'protect' and a victim model, will generate poisoned data with the following properties:

(1) A victim model trained on the poison will misclassify the protected class.
(2) The poison will not appear to belong to the protected class. A desideratum in data poisoning is the continued functioning of the poison even if the attacker does not supply the poison labels. Shafahi et al.'s PoisonFrogs [14], for instance, functions even if the victim provides labels for the poisoned examples.
(3) The poison will be correctly classified as the class it purports to be. Part of the strategy at play is that the victim model's performance should not appear to degrade; we therefore wish it to only misclassify the protected class – its performance for other classes should be unimpeded, and that includes the classes containing poisoned examples.

If applied correctly, this tactic could 'shield' certain classes one wishes to protect, while incurring an undetectably small performance degradation in the model – especially if the number of classes is large.

How, then, are we to devise such poisons? From our use case, we can see a desirable artifact: an ML metalearner (after Huang et al. [9]) that can, given examples of a class, devise poison to shield that class, given a target model or models (about which it need know nothing – that is, it is a black-box attack).

2 METHOD

2.1 Dataset

This work's focus is on developing a metalearner for poison generation, and not on developing victim models against which the poisoner can be trained; hence, it behooves us to select a domain and

data with many available pre-trained state-of-the-art classification models.

Image recognition is ripe for this application; not only because it is a fecund domain for Data Poisoning, but also because the complexity of an image recognizer network lies primarily in its feature extractors, which we can cheaply transfer to new victim models. Additionally, according to Mayerhofer and Mayer [12], CNN-based image classification is a domain with suggested resilience to poisoning, and thus presents a challenge.

Most publicly available library-implemented SOTA image classifiers are pre-trained on the ILSVRC 2012 ("ImageNet") dataset [13]; we therefore acquired a copy of ImageNet. Our preprocessing needs were minimal.

Due to time constraints we elected to use two pretrained classifiers, VGG16 [15] and ResNet50 [8]. Our victim models were formed by removing the fully-connected terminal layers of each and replacing them with a fresh fully-connected layer.

Thanks to Tensorflow's flexibility and usability considerations, implementing the nontrivial training pipeline took most of the month allotted to the project (see 4.2). In order to keep the training process at a feasible timescale, therefore, we downsampled the training set to 12729 images across 10 classes.

2.2 Training Process Outline

Figure 1 shows the outline of the training loop dataflow. The dataset was split 80:20 training:validation, and images were grouped into "super-batches" of 1000 apiece.

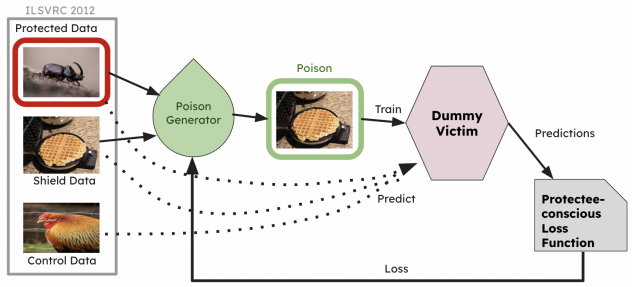

Figure 1: Summary of the training loop.

In each training step, the super-batch was divided into three: a "Protected Class", which the victim is intended to misclassify, a "Shield Class", whose perturbations form the poison, and a Control class. Pairs of protected and shield images were passed to the poison generator, which produced a single poison image per pair (details in 2.3). The resulting super-batch of poison was divided into sub-batches of 10 images apiece.

The victim model is represented in the learning pipeline by a 'dummy victim', which is a transfer-learning-headless version of the target architecture with a fully-connected terminal classification layer whose weights are reset at each iteration of the full training loop. This dummy model was trained on the poison generated (where each poison is labeled with the class of the shield datum from which it was generated), with weights updating every sub-batch.

The resulting trained victim model was then used to classify all data in the super-batch. Losses for data in the "Shield" and "Control" sets were computed with standard Categorical Crossentropy ($L_{CE} = -\sum_{i=1}^{n} t_i \ln(p_i)$ for truth values $\mathbf{t}$ and softmax probabilities $\mathbf{p}$ over n classes). As we wished the model to misclassify the target class, however, losses for target data were computed with a horizontally flipped crossentropy $L_{\overline{CE}} = -\sum_{i=1}^{n} t_i \ln(-p_i + 1)$. This transformation was chosen as perfect confidence in the protected class would result in as much loss as no confidence in an unprotected class, while total lack of confidence in the protected class results in zero loss. Total loss was computed as

$$L_{total} = \frac{\frac{L_{CE}(Shield)+L_{CE}(Control)}{2} + (L_{\overline{CE}}(Protected)t)}{2}$$

where t is chosen to adjust the relative weight of the loss of the protected class. In practice we have yet to determine the effect of this weight on behaviour.

The computed total loss was then used to update the weights of the poison generator, thus completing the training cycle.

2.3 Poison Generator Architecture

Figure 2 shows the network architecture for the poison generator subsystem. As discussed above, it takes one image to 'protect' and one image to use as a 'shield', and produces one poisoned image.

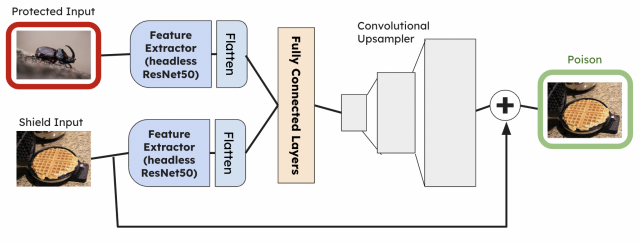

Figure 2: Network structure for the prototype poison generator.

We elected to employ a residual model, in which the poisoning function approximated is added to the shield image, resulting in a perturbed shield. This is consistent with our requirement that the poison should be undetectable; if the perturbation is learned correctly, the poison can be labeled as the shield class and should be classified by the model as the shield class, with no degradation in accuracy.

Pursuant to our aim that this project should involve as little wheel-reinvention as possible, we employ a pre-trained ResNet50 to perform feature extraction. The extractor's weights are not updated as part of the training process.

Extracted features are flattened, then fed into a series of fully-connected layers which form the core of the poison generator. In our final incarnation these consisted of one 2048-neuron layer and one 768-neuron layer, the latter chosen as it could be reshaped into a 16 pixel × 16 pixel × 3 channel image output.

The output from the connected layers is then fed into an upsampler that returns it to the dimensions of the input data. Our approach

uses two deconvolutional layers. The resulting perturbation is then added to the shield input to produce the final poison.

3 RESULTS

3.1 Target Architecture: ResNet50

The model was fitted with a victim model derived from ResNet50 and trained for 10 epochs. The resulting intra-test and validation accuracies are shown in figure 3. Training accuracies for the shield/control and for the target appear to diverge slightly after epoch 5, but the margin is too small to be certain of any trend. Additionally, it is not reflected in the validation, which shows only a slight degradation in overall accuracy, and is therefore probably merely due to overfitting.

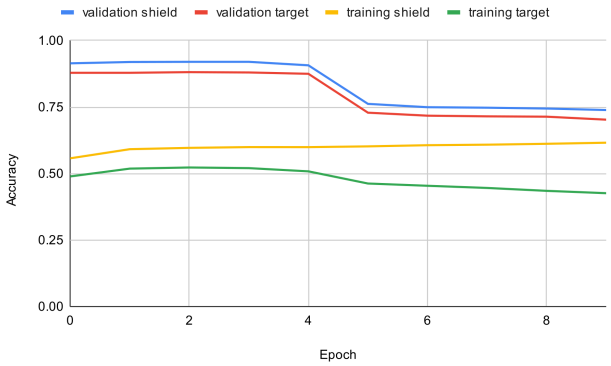

Figure 3: Training and validation accuracy for a victim ResNet50 per epoch.

3.2 Target Architecture: VGG16

The same procedure as above was repeated with a VGG-16 target model. As can be seen in figure 4, equally poor performance was obtained; additionally, the VGG16 architecture as a whole appears far less performant in general than the ResNet on these small data. Of interest is that the validation performance degrades in a quasi-linear fashion, as opposed to the ResNet performance, in which we observe a steep drop between slow declines or plateaus.

4 CONCLUSIONS

4.1 Analysis of Results

The results above do not show the performance sought from the model. The low general accuracy is acceptable, as the victim models were only shown a small amount of poison; we had hoped to observe a favourable trend in relative accuracy. This trend is somewhat present in the training performance; one can observe the target training accuracy sinking slowly, and the shield training accuracy rising slowly, but as the trend is not at all mirrored in the validation performance it is likely an artifact of overfitting.

The graphs also show degrading validation performance for both classes over the epochs. This may suggest that the latter-epoch poisons increasingly degrade the overall performance of the model; that they 'strengthen' with training; why this may be is unclear.

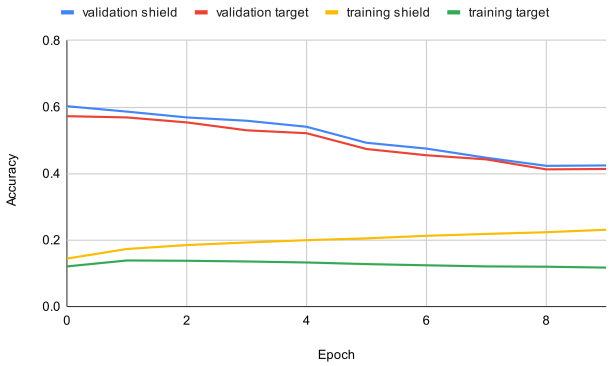

Figure 4: Training and validation accuracy for a victim VGG16 per epoch.

4.2 Challenges

The primary difficulty encountered while implementing this project was the lack of support in TensorFlow for unorthodox training procedures. Part of the feed-forward behaviour of the overall poisoner model consists of training the dummy victim (as the loss cannot be computed until after the dummy victim has both been trained on poison and has predicted on data); TensorFlow's GradientTape Autodifferentiator cannot trace data-flow through the training of another model, and so was unable to compute gradients for the poisoner.

We circumvented this weakness by exploiting the determinism in our model: at each training step, the poisoner generates poison, on which the victim is trained. The poisoner then generates the *same* poison again, which is then fed through the dummy victim for prediction, thus maintaining a data flow from source to loss and allowing the GradientTape to function.

4.3 Potential Improvements

Several possible avenues present themselves for further work in order to make our architecture efficacious:

(1) *U-NET upsampling*. Currently our poisoner network generates poison by extracting features, then feeding them to a feed-forward block, then upsampling with deconvolutions. The deconvolution layers are therefore operating 'in the blind' and must learn features that are general to all classes of poison. If our architecture had through-lines of data from the feature extractor to the upsampler (as in a U-Net design), the upsampling process would be data-gnostic and perhaps more performant as a result.
(2) *Alternative Hyperparameters*. Due to time constraints our exploration of the hyperparameter space for the extant model architecture was limited. Certain untested values of the following might prove fruitful: numbers of nodes in the dense feed-forward layers, number of said layers, number of deconvolutional upsampling layers, and minimal 'bottleneck' dimension. Additionally, adjustments to several network components, such as alternative pretrained feature

extractors (VGG16, say) or adjustments to the final residual sum step, might serve.

(3) *Alternative Architecture.* Our choice of a residual image-feature-based architecture for the poisoner was based on accepted wisdom of the suitability of that architecture for the image domain; however, alternative viable architectures exist, such as transformers. Fitting the model with an attention mechanism might prove advantageous.

(4) *Full Training Schedule.* In order to finish model training in the allotted time we downsampled the ImageNet dataset (see end of 2.1). Complex Deep Learning problems require larger quantities of data to work, and so our model might benefit from being trained on the entire ImageNet dataset.

4.4 Future Work

Although the model is at present nonfunctional, if it were there would remain some practical considerations.

Firstly, the images are preprocessed into the input format accepted by the pretrained classifier model; in both ResNet50 and VGG16 cases this means mean-centering with respect to ImageNet and colour channel swapping. Residual poisons derived therefrom thus require a deprocessing step to return them to the domain of plausible inputs. Such a step may compromise the poison's performance.

Secondly, the residual architecture was chosen partly to improve the likelihood that the poisoned images would be unrecognizable to human observers. No attempt was made to verify human-undetectability, however, partly due to the lack of deprocessing mentioned above, and partly due to the subjective nature of such analysis.

Thirdly, we made no provision for data sanitization techniques that have demonstrated anti-poison efficiency (for instance Borgnia et al.'s data augmentation schemes [2] or Aladag et al.'s Initial Autoencoder step [1]). In section 4.3 above we discuss alternative architectures for the poisoner; some architectures may perform better against specific defences than others.

REFERENCES

[1] Merve Aladag, Ferhat Ozgur Catak, and Ensar Gul. 2019. Preventing Data Poisoning Attacks By Using Generative Models. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)*. 1–5. https://doi.org/10.1109/UBMYK48245.2019.8965459

[2] Eitan Borgnia, Valeriia Cherepanova, Liam Fowl, Amin Ghiasi, Jonas Geiping, Micah Goldblum, Tom Goldstein, and Arjun Gupta. 2021. Strong Data Augmentation Sanitizes Poisoning and Backdoor Attacks Without an Accuracy Tradeoff. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 3855–3859. https://doi.org/10.1109/ICASSP39728.2021.9414862 ISSN: 2379-190X.

[3] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A. Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. 2023. Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning. *Comput. Surveys* (March 2023). https://doi.org/10.1145/3585385 Just Accepted.

[4] Minghong Fang, Minghao Sun, Qi Li, Neil Zhenqiang Gong, Jin Tian, and Jia Liu. 2021. Data Poisoning Attacks and Defenses to Crowdsourcing Systems. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, New York, NY, USA, 969–980. https://doi.org/10.1145/3442381.3450066

[5] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. 2020. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. https://doi.org/10.48550/arXiv.1902.06531 arXiv:1902.06531 [cs].

[6] Jonas Geiping, Liam Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. 2021. Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching. https://doi.org/10.48550/arXiv.2009.02276 arXiv:2009.02276 [cs].

[7] Jonas Geiping, Liam Fowl, Gowthami Somepalli, Micah Goldblum, Michael Moeller, and Tom Goldstein. 2022. What Doesn't Kill You Makes You Robust(er): How to Adversarially Train against Data Poisoning. https://doi.org/10.48550/arXiv.2102.13624 arXiv:2102.13624 [cs].

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[9] W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. 2020. MetaPoison: Practical General-purpose Clean-label Data Poisoning. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 12080–12091. https://proceedings.neurips.cc/paper/2020/hash/8ce6fc704072e351679ac97d4a985574-Abstract.html

[10] Matthew Jagielski, Giorgio Severi, Niklas Pousette Harger, and Alina Oprea. 2021. Subpopulation Data Poisoning Attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3104–3122. https://doi.org/10.1145/3460120.3485368

[11] Zeyan Liu, Fengjun Li, Zhu Li, and Bo Luo. 2022. LoneNeuron: A Highly-Effective Feature-Domain Neural Trojan Using Invisible and Polymorphic Watermarks. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2129–2143. https://doi.org/10.1145/3548606.3560678

[12] Robin Mayerhofer and Rudolf Mayer. 2022. Poisoning Attacks against Feature-Based Image Classification. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy (CODASPY '22)*. Association for Computing Machinery, New York, NY, USA, 358–360. https://doi.org/10.1145/3508398.3519363

[13] 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. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (Dec. 2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[14] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. https://doi.org/10.48550/arXiv.1804.00792 arXiv:1804.00792 [cs, stat].

[15] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[16] Zhensu Sun, Xiaoning Du, Fu Song, Mingze Ni, and Li Li. 2022. CoProtector: Protect Open-Source Code against Unauthorized Training Usage with Data Poisoning. In *Proceedings of the ACM Web Conference 2022 (WWW '22)*. Association for Computing Machinery, New York, NY, USA, 652–660. https://doi.org/10.1145/3485447.3512225

[17] Yuxi Zhao, Xiaowen Gong, Fuhong Lin, and Xu Chen. 2021. Data Poisoning Attacks and Defenses in Dynamic Crowdsourcing with Online Data Quality Learning. *IEEE Transactions on Mobile Computing* (2021), 1–1. https://doi.org/10.1109/TMC.2021.3133365 Conference Name: IEEE Transactions on Mobile Computing.