Simple, Attack-Agnostic Defense Against Targeted Training Set Attacks Using Cosine Similarity

Zayd Hammoudeh [1] **Daniel Lowd** [1]

Abstract

Targeted training set attacks inject adversarially perturbed instances into the training set to cause the trained model to behave aberrantly on specific test instances. As a defense, we propose to identify the most influential training instances (likely to be attacks) and the most influenced test instances (likely to be targets). Among prior influence estimation methods, TracIn shows the most promise but still performs poorly. We therefore propose a cosine similarity influence estimator, CosIn, which improves upon TracIn by focusing on gradient direction over magnitude. In experiments on vision, NLP, and speech domains, CosIn identifies up to 100% of adversarial instances in poisoning and backdoor training attacks. Our source code is available at https://github.com/ZaydH/cosin.

1. Introduction

In *targeted training set attacks*, an attacker corrupts the training data used by a machine learning system in order to influence the learned model and its prediction on a particular input (or set of inputs). For example, a business may wish to influence a sentiment classifier so that their product is always viewed positively while a competitor's is always viewed negatively. Recent work has shown that deep learning systems can be manipulated like this using only a few examples (Shafahi et al., 2018; Wallace et al., 2021).

Defending against such attacks remains a challenge. Certifiably robust learning approaches (Steinhardt et al., 2017; Jia et al., 2021; Levine & Feizi, 2021; Weber et al., 2021) prevent a small number of examples having an outsized influence on the model, which helps avoid training set attacks but also interferes with learning legitimate yet rare events.

[1]Department of Computer & Information Science, University of Oregon, Eugene, OR, USA. Correspondence to: Zayd Hammoudeh <zayd@cs.uoregon.edu>.

Presented at the ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning., Copyright 2021 by the author(s).

Another approach is to directly identify the manipulated examples (Chen et al., 2019; Gao et al., 2019; Wang et al., 2019; Peri et al., 2020), but prior work is typically limited to detecting specific attacks or working in specific domains.

Since training set attacks are only effective when they influence the model, a natural strategy is to identify the most influential training examples. If an attack's target is known (or can be determined), the search narrows to examples that influence that target. Several influence estimation methods exist, including influence functions (Koh & Liang, 2017), representer point (Yeh et al., 2018), and methods that aggregate gradients throughout the training process (Hara et al., 2019; Pruthi et al., 2020; Chen et al., 2021).

In this paper, we explore how influence estimation methods can be adapted to find targeted training set attacks. Our contributions are as follows:

1. We compare influence functions, representer points, and training gradient methods and find the latter much more effectively identifies targeted training set attacks.
2. We propose a modified training set influence estimator, CosIn, which (1) scores all examples in each training checkpoint, instead of just those in the associated minibatch(es); and (2) normalizes by the gradient magnitudes when computing dot products, so influence is determined by gradient direction or alignment – not magnitude.
3. We evaluate CosIn on datasets from vision, NLP, and speech with poison and backdoor attacks and find that CosIn vastly outperforms previous methods for influence estimation and poison detection.

2. Problem Formulation

Threat Model The attacker adds a small set of adversarial training examples, $\mathcal{D}_{\text{adv}}$, to clean examples $\mathcal{D}_{\text{cl}}$, forming the full training dataset, $\mathcal{D}_{\text{tr}} := \mathcal{D}_{\text{adv}} \cup \mathcal{D}_{\text{cl}}$. We assume training uses an iterative, first-order optimization algorithm, denoted $\mathcal{A}$, (e.g., gradient descent, Adam) that runs for T iterations starting from initial parameters θ_0. In each iteration $t \in [T]$, $\mathcal{A}$ uses the loss gradient of random minibatch $\mathcal{B}_t \subset \mathcal{D}_{\text{tr}}$ to update model parameters θ_{t-1} to θ_t.

Attacker's Objective & Knowledge The attacker's objective is to influence the training process such that for some *target* test example, $z_{\text{targ}} := (x_{\text{targ}}, y_{\text{targ}})$,[1] the learner mispredicts the instance as having label $y_{\text{adv}} \neq y_{\text{targ}}$. Backdoor attacks insert the same, fixed *adversarial trigger* (e.g., change one specific pixel to maximum value) to all examples in $\mathcal{D}_{\text{adv}}$ and change these perturbed examples' labels to y_{adv}. The same adversarial trigger is also added to the target example's feature vector x_{targ}. Data poisoning only perturbs training examples with the target example's features, x_{targ}, remaining pristine. *Clean-label* poisoning attacks leave labels unchanged when crafting $\mathcal{D}_{\text{adv}}$ from a set of seed instances.

Attacker knowledge of the learning environment varies. Inline with previous work (Zhu et al., 2019; Huang et al., 2020; Wallace et al., 2021), poisoning attacks a specific pretrained model – excluding randomly initialized linear layers. The backdoor attacks are model-agnostic and here are trained from scratch as in (Liu et al., 2018; Weber et al., 2021). The attacker never knows the training minibatch sequence.

Our Objective Let $\mathcal{T} \subset \{1, \ldots, T\}$ denote a subset of the training iterations where $|\mathcal{T}| \ll T$, and let $\mathcal{P} := \{(\eta_t, \theta_{t-1}) : t \in \mathcal{T}\}$ be serialized, periodic training parameters where η_t is iteration t's learning rate and θ_{t-1} is the iteration's starting model parameters. Provided $\mathcal{P}$ and test example $z_{\text{te}} := (x_{\text{te}}, y_{\text{te}})$, our goals are two-fold: (1) *detect* if $z_{\text{te}} = z_{\text{targ}}$, and if so, (2) use z_{te} to *identify* $\mathcal{D}_{\text{adv}}$. We then retrain to render the attack ineffective.

3. Influence Estimation for Training Set Attacks

Pruthi et al. (2020) propose TracIn – a general-purpose influence estimation method; it estimates influence by "tracing" the training procedure, adding up the impact of each example each time its gradient is used to update the model. This can be approximated as a dot product between training and target gradients, and further approximated by using periodic model checkpoints instead of every intermediate model seen during training. See suppl. Alg. 3 for TracIn's pseudocode.

In preliminary experiments, we found that TracIn better identified training set attacks than other influence estimation methods, e.g., influence functions (Koh & Liang, 2017) and representer point methods (Yeh et al., 2018). However, TracIn has two limitations that make it less effective for our objective[2]. Our proposed method, COSIN (Alg. 1)[3], addresses these limitations by:

Algorithm 1 COSIN influence estimation

Input: Periodic training parameter checkpoints $\mathcal{P}$, training set $\mathcal{D}_{\text{tr}}$, and target example z_{targ}
Output: COSIN influence values $\mathbf{v}$

$\mathbf{v} \leftarrow \vec{0}$ ▷ Initialize influence
for each $(\eta_t, \theta_{t-1}) \in \mathcal{P}$ **do**
  $g_{\text{targ}} \leftarrow \nabla_\theta \ell\left(z_{\text{targ}}; \theta_{t-1}\right)$
  for each $z_i \in \mathcal{D}_{\text{tr}}$ **do** ▷ All examples
    $g_i \leftarrow \nabla_\theta \ell\left(z_i; \theta_{t-1}\right)$
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \eta_t \frac{\langle g_{\text{targ}}, g_i \rangle}{\|g_{\text{targ}}\|_2 \|g_i\|_2}$ ▷ Normalized
return $\mathbf{v}$

1. In TracIn, an example's influence depends on the randomly chosen batches to which it was assigned, so copies of the same example will end up with different influence. Checkpoint TracIn avoids this by performing one update per epoch, using all examples, but the approximation quality will be poor if model parameters change substantially within an epoch. *Therefore,* COSIN *considers every example at each checkpoint and supports "subepoch" checkpoints at any frequency within an epoch*[4].
2. TracIn uses un-normalized gradients, giving more influence to high-loss examples that have larger gradients, even though such examples align poorly with target, z_{targ}. However, we observed that poison examples rarely have the largest gradient magnitudes (see Figs. 1c & 1f). *Therefore,* COSIN *normalizes gradients by their magnitudes, i.e., uses cosine similarity, when computing influence.*

The first change affects how we sum influence over multiple training examples and optimization steps. The latter change is more significant, because it changes how each element's influence is computed. We refer to this modified method as COSIN, since it estimates the target influence as the cosine similarity averaged over the course of training.

Recent poisoning methods, e.g., (Huang et al., 2020; Wallace et al., 2021), track crafted poison's gradients during simulated training. Hence, poison's influence may be best observed throughout or at very specific points in training.

As an intuition, note that *all* training set attacks only influence the model via gradients. By focusing on gradients, COSIN is *attack agnostic*, making COSIN applicable to all training set attacks – even those not yet known/devised.

Determining the Target Example

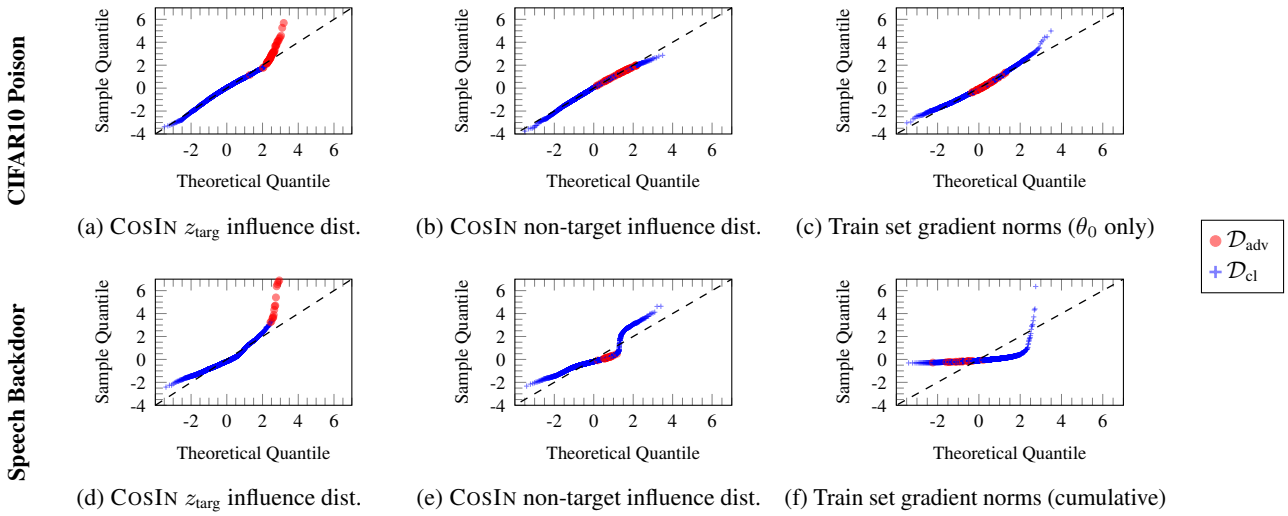
COSIN and the other influence estimation methods compute each training example's influence on a specific target. Gen-

[1] We consider a single target example for simplicity and w.l.o.g.

[2] SGD-Cleanse (Hara et al., 2019) operates under similar principles as TracIn. This work builds on TracIn since SGD-Cleanse is designed for untargeted data cleaning.

[3] COSIN's and TracIn's training algorithm is in suppl. Alg. 2.

[4] We use the term "m subepoch checkpointing" to denote m evenly-spaced checkpoints within each epoch.



(a) COSIN z_{targ} influence dist. (b) COSIN non-target influence dist. (c) Train set gradient norms (θ_0 only)

(d) COSIN z_{targ} influence dist. (e) COSIN non-target influence dist. (f) Train set gradient norms (cumulative)

Figure 1. Quantile-quantile (QQ) plots with respect to a theoretical normal distribution for COSIN influence on CIFAR10 convex polytope poisoning (Zhu et al., 2019) (label dog → bird) and speech recognition backdoor (Liu et al., 2018) (digit 5 → 6) attacks. COSIN used only θ_0 for CIFAR10 and 10 subepoch checkpointing for speech. COSIN's influence distribution over training set, $\mathcal{D}_{tr}$, had much heavier upper tails when evaluating target example, z_{targ}, (Fig. 1a & 1d) than for a non-target example (Fig. 1b & 1e). Training example gradient norms were poorly correlated with whether the training example was adversarial (Fig. 1c & 1f).

erally, the defender does not know the target. Thus, our ability to find and remove poison may rely on whether we can first identify the target of any potential poison.

Empirically, we find that training set attack targets tend to have many exceptionally influential examples, namely $\mathcal{D}_{adv}$, while non-targets do not (see Figure 1). We turn this intuition into a quantitative method by measuring the one-sided outliers for each potential target example, with real targets tending to have an influence distribution with a heavy upper tail, including an unusually high number of training examples with influence several standard deviations above the mean. To be more statistically robust to high influence outliers, median is used in place of mean and Rousseeuw & Croux's (1993) Q estimator instead of standard deviation.

4. Evaluation

We evaluate our method on three attacks – data poisoning in vision and text, and backdoor attacks in speech recognition. Our two tasks are to identify the target instance z_{targ} and the attack instances $\mathcal{D}_{adv}$. Because poison is relatively rare, we measure performance using area under the precision-recall curve (AUPRC), which quantifies how well the actual attacks rank relative to clean training instances. See suppl. Sections B and C for more details and additional experiments, including all speech recognition results.

Baselines We compare to TracIn, influence functions, and representer points, all of which estimate influence and hence are comparable to COSIN. Section 4.1 also considers clean-label poison defense Deep k-NN (Peri et al., 2020). k-NN yields only a label, not a score. To be compatible with AUPRC, we modified Deep k-NN to rank each training example by the difference between the size of the neighborhood's plurality class and the number of neighborhood instances that share the corresponding example's label.

4.1. Defending Against Clean-Label Vision Poisoning

Zhu et al.'s (2019) targeted, clean-label attack crafts a set of poisons by forming a *convex polytope* around the target's feature representation. Similar to Zhu et al.'s experimental setting, our evaluation follows the pretrain then fine-tune paradigm on CIFAR10 (Krizhevsky et al., 2014).

In each trial, ResNet9 was pretrained from scratch using half the classes (plane, car, ship, truck, & horse). z_{targ} was then randomly selected from the test examples labeled y_{targ}, and then $|\mathcal{D}_{adv}| = 50$ gray-box poisons (0.2% of $\mathcal{D}_{tr}$) were crafted from seed training examples labeled y_{adv}. Lastly, the pretrained network (with the linear layer's parameters randomly set) was fine-tuned using $\mathcal{D}_{adv}$ and the unused training examples labeled bird, cat, deer, dog, and frog. As in (Shafahi et al., 2018; Huang et al., 2020), the target/poison label pairs are dog ↔ bird and deer ↔ frog.

Poison Identification Figure 2 visualizes COSIN's poison identification AUPRC (both when using just θ_0 and with 5 subepoch checkpointing) versus the baselines. For all class pairs, COSIN identified vastly more poison. While COSIN using only θ_0 has comparable mean performance to subepoch checkpointing, it also has higher variance (see suppl. Table 9). This lower variance comes at the expense

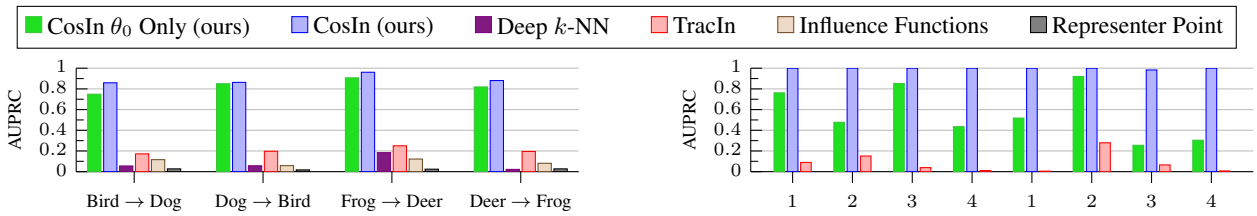


Figure 2. Vision Poison Identification: Mean poison identification AUPRC with 50 poison across >50 trials for four CIFAR10 class pairs. X-axis denotes the target class → to poison class resp. Both versions of CosIn vastly outperformed all baselines for all class pairs. See Table 9 for full numerical results inc. variance.

Figure 3. Natural Language Poison Identification: Mean poison identification AUPRC with 50 poison across 10 trials for 4 pos. and 4 neg. sentiment SST-2 reviews. Influence func. & representer pt. both had AUPRC <0.03 and are excluded above. CosIn perfectly identified all poison in 78/80 trials. See Table 14 for full results.

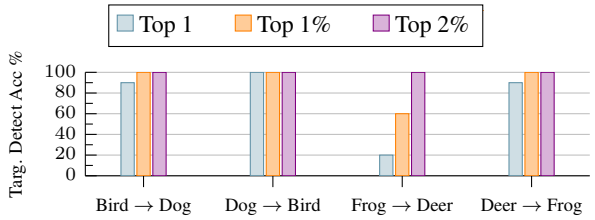


Figure 4. Vision Poison Target Detection: CosIn's (using only θ_0) top-K target, z_{targ}, detection accuracy across 10 trials for each y_{targ} with 350 clean test examples per trial. The cutoff threshold was $5Q$ above the median CosIn influence. All values are percentages.

of longer execution time (see suppl. Table 15).

Poison Target Detection Figure 4 shows CosIn's (using only θ_0) top-K z_{targ} detection accuracy across 10 trials for each class pair. In each trial, the training set's CosIn influence was measured separately for z_{targ} and 350 clean test instances (all labeled y_{targ}). CosIn influence's upper-tail heaviness (i.e., number of training examples with influence $\geq 5Q$ above the median) was calculated and the test instances/z_{targ} ordered accordingly. For dog → bird, z_{targ} always had the heaviest upper tail (i.e., had perfect top-1 accuracy). For bird → dog and deer → frog, z_{targ} was top-1 in 9/10 trials and top-2 in 1/10. CosIn always had perfect detection accuracy at top-2%.

Poison Removal In these experiments, a poisoned vision model is trained as described above. Next CosIn (using only θ_0) was performed over $\mathcal{D}_{\text{tr}}$, and any training examples with CosIn influence a specified multiplier of Q above the median were removed. A new, fine-tuned model was then trained starting from θ_0.

As shown in suppl. Table 11, CosIn (using only θ_0) always successfully blocked the attack against bird → dog and frog → deer across trials while removing as little as 0.01% of $\mathcal{D}_{\text{cl}}$. CosIn similar defended 9/10 attacks against dog → bird and deer → frog while similarly removing 0.01% of $\mathcal{D}_{\text{cl}}$. In contrast, Deep k-NN may remove up to 4.3% of $\mathcal{D}_{\text{cl}}$ on average.

Remark CosIn with 5 subepoch checkpointing was the method that best identified poison (see Figure 2), and we expect it would also have better target detection and poison removal performance than using just θ_0 as in Figure 4.

4.2. Defending Against Natural Language Poisoning

Wallace et al. (2021) construct natural language poison using the traditional poisoning bilevel optimization (Biggio et al., 2012; Muñoz-González et al., 2017). To make the computation tractable, Wallace et al. approximate the inner minimizer using second-order gradients similar to (Finn et al., 2017; Wang et al., 2018; Huang et al., 2020). Wallace et al.'s method[5] initializes each poison instance from a seed phrase, and tokens are iteratively replaced with alternates that align well with the poison example's gradient.

Like Wallace et al., our experiments attacked sentiment analysis on the Stanford Sentiment Treebank v2 (SST-2) dataset (Socher et al., 2013) – specifically on RoBERTa$_{\text{BASE}}$ (Liu et al., 2020), which has 125M parameters. We targeted 8 short, randomly-selected reviews (4 pos. & 4 neg.) and generated $|\mathcal{D}_{\text{adv}}| = 50$ new poison in each trial.

Poison Identification Figure 3 compares each method's mean performance across 10 trials for each of the 8 reviews. In 78/80 trials, CosIn (with 3 subepoch checkpointing) perfectly identified all poison – vastly better than the baselines. Even CosIn over just θ_0 well outperformed the baselines.

Observe that the gap between CosIn using only θ_0 versus subepoch checkpointing is much larger than it was for CIFAR10 in Figure 2. This demonstrates the importance of considering all epochs. If θ_0 is not pretrained (as with suppl. Section C.2's speech recognition backdoor attack), extracting significant information from θ_0 may not be possible.

[5]Text poison was generated using Wallace et al.'s original implementation which was provided via personal correspondence.

Acknowledgements

This work was supported by a grant from the Air Force Research Laboratory and the Defense Advanced Research Projects Agency (DARPA) — agreement number FA8750-16-C-0166, subcontract K001892-00-S05, as well as a second grant from DARPA, agreement number HR00112090135. This work benefited from access to the University of Oregon high performance computer, Talapas.

References

Basu, S., Pope, P., and Feizi, S. Influence functions in deep learning are fragile. In *Proceedings of the 9th International Conference on Learning Representations*, ICLR'21, 2021.

Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning*, ICML'12, 2012.

Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., and Srivastava, B. Detecting backdoor attacks on deep neural networks by activation clustering. In *Proceedings of the AAAI Workshop on Artificial Intelligence Safety*, SafeAI'19, 2019.

Chen, Y., Li, B., Yu, H., Wu, P., and Miao, C. HyDRA: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, AAAI'21, 2021.

Coleman, C. A., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. DAWNBench: An end-to-end deep learning benchmark and competition. In *Proceedings of the 2017 NeurIPS Workshop on Machine Learning Systems*, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, 2017.

Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D. C., and Nepal, S. STRIP: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC'19, 2019.

Hara, S., Nitanda, A., and Maehara, T. Data cleansing for models trained with SGD. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, NeurIPS'19, 2019.

Huang, W. R., Geiping, J., Fowl, L., Taylor, G., and Goldstein, T. MetaPoison: Practical general-purpose clean-label data poisoning. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, NeurIPS'20, 2020.

Jia, J., Cao, X., and Gong, N. Z. Intrinsic certified robustness of bagging against data poisoning attacks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, AAAI'21, 2021.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, 2017.

Krizhevsky, A., Nair, V., and Hinton, G. The CIFAR-10 dataset, 2014.

Levine, A. and Feizi, S. Deep partition aggregation: Provable defenses against general poisoning attacks. In *Proceedings of the 9rd International Conference on Learning Representations*, ICLR'21, 2021.

Liu, Y., Ma, S., Aafer, Y., Lee, W.-C., Zhai, J., Wang, W., and Zhang, X. Trojaning attack on neural networks. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, NDSS'18, 2018.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A robustly optimized BERT pretraining approach. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR'20, 2020.

Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E. C., and Roli, F. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Page, D. How to train your ResNet, May 2020. URL https://myrtle.ai/learn/how-to-train-your-resnet/.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, NeurIPS'19, 2019.

Pearlmutter, B. A. Fast exact multiplication by the Hessian. *Neural Computation*, 6:147–160, 1994.

Peri, N., Gupta, N., Huang, W. R., Fowl, L., Zhu, C., Feizi, S., Goldstein, T., and Dickerson, J. P. Deep k-NN defense against clean-label data poisoning attacks. In *Proceedings of the ECCV Workshop on Adversarial Robustness in the Real World*, AROW'20, 2020.

Pruthi, G., Liu, F., Kale, S., and Sundararajan, M. Estimating training data influence by tracing gradient descent. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, NeurIPS'20, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners, 2019.

Rousseeuw, P. and Croux, C. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 1993.

Shafahi, A., Huang, W. R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, NeurIPS'18, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 8th Conference on Empirical Methods in Natural Language Processing*, EMNLP'13, 2013.

Steinhardt, J., Koh, P. W., and Liang, P. Certified defenses for data poisoning attacks. In *Proceedings of the 30st Conference on Neural Information Processing Systems*, NeurIPS'17, 2017.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2013.

Wallace, E., Zhao, T. Z., Feng, S., and Singh, S. Concealed data poisoning attacks on NLP models. In *North American Chapter of the Association for Computational Linguistics*, NAACL'21, 2021.

Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proceedings of the 40th IEEE Symposium on Security and Privacy*, 2019.

Wang, T., Zhu, J., Torralba, A., and Efros, A. A. Dataset distillation, 2018.

Weber, M., Xu, X., Karlaš, B., Zhang, C., and Li, B. RAB: Provable robustness against backdoor attacks, 2021.

Yeh, C., Kim, J. S., Yen, I. E., and Ravikumar, P. Representer point selection for explaining deep neural networks. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, NeurIPS'18, 2018.

Zhu, C., Huang, W. R., Shafahi, A., Li, H., Taylor, G., Studer, C., and Goldstein, T. Transferable clean-label poisoning attacks on deep neural nets. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, 2019.

Simple, Attack-Agnostic Defense Against Targeted Training Set Attacks Using Cosine Similarity

Supplemental Materials

A. Additional Algorithms

Algorithm 2 shows the minor modifications made to standard minibatch training to accomodate CosIn and TracIn.

Algorithm 2 CosIn & TracIn training phase

Input: Optimization algorithm $\mathcal{A}$, initial parameters θ_0, training set $\mathcal{D}_{\text{tr}}$, learning rate η, iteration subset $\mathcal{T}$, iteration count T, and initial parameters θ_0
Output: Training parameters $\mathcal{P}$

$\mathcal{P} \leftarrow \emptyset$
for $t \leftarrow 1$ **to** T **do**
  if $t \in \mathcal{T}$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\eta_t, \theta_{t-1})\}$
  $\mathcal{B}_t \sim \mathcal{D}_{\text{tr}}$
  $\theta_t \leftarrow \mathcal{A}(\eta_t, \theta_{t-1}, \mathcal{B}_t)$
return $\mathcal{P}$

Algorithm 3 details TracIn's basic algorithm for minibatch gradient descent. Note that TracIn always includes the first iteration in its set of serialized parameters, i.e., $1 \in \mathcal{T}$, which is why Line 3's `if` statement can be inside Line 2's `for` loop.

Algorithm 3 TracIn inference phase

Input: Training parameters $\mathcal{P}$, iteration subset $\mathcal{T}$, batch sequence $\mathcal{B}_1, \ldots, \mathcal{B}_T$, and target example z_{targ}
Output: TracIn influence values $\mathbf{v}$

$\mathbf{v} \leftarrow \vec{0}$ ▷ Initialize influence
for $t \leftarrow 1$ **to** T **do**
  if $t \in \mathcal{T}$ **then**
    $(\eta, \theta) \leftarrow \mathcal{P}[t]$ ▷ Equiv. to (η_t, θ_{t-1})
    $g_{\text{targ}} \leftarrow \nabla_\theta \ell\left(z_{\text{targ}}; \theta\right)$
  for each $z_i \in \mathcal{B}_t$ **do** ▷ Batch examples
    $g_i \leftarrow \nabla_\theta \ell\left(z_i; \theta\right)$
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \eta \langle g_{\text{targ}}, g_i \rangle$ ▷ Unnormalized
return $\mathbf{v}$

B. Evaluation Setup

This section details the evaluation setup used in Section 4's experiments, including dataset specifics, hyperparameters, and the neural network architectures.

Our source code can be downloaded from `https://github.com/ZaydH/cosin`. All experiments used the PyTorch automatic differentiation framework (Paszke et al., 2019) and were tested with Python 3.6.5. Wallace et al.'s (2021) sentiment analysis data poisoning source code will be published by its authors at `https://github.com/Eric-Wallace/data-poisoning`.

B.1. Dataset Sizes

Table 1 details the dataset sizes used to train all evaluated models.

Table 1. Dataset sizes

Dataset	Attack	# Train	# Test
CIFAR10 (Krizhevsky et al., 2014)	Poison	25,000	5,000
SST-2[6] (Socher et al., 2013)	Poison	67,349	N/A
Speech (Liu et al., 2018)	Backdoor	3,000[7]	1,184

Liu et al.'s (2018) speech backdoor dataset includes training and test examples with their associated adversarial trigger. We use this adversarial dataset unchanged. Table 2 details $|\mathcal{D}_{\text{adv}}|$ (i.e., adversarial training set size) for each speech digit pair.

Table 2. Number of backdoor training examples for each speech backdoor digit pair

Digit Pair	$0 \to 1$	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	$6 \to 7$	$7 \to 8$	$8 \to 9$	$8 \to 9$
$\lvert\mathcal{D}_{\text{adv}}\rvert$	16	17	14	14	16	18	16	16	12	11

B.2. Hyperparameters

B.2.1. Model Training

Table 3 enumerates the hyperparameters used when training the models analyzed in Sections 4 and C.

Table 3. Model training hyperparameter settings

	Poison		Backdoor
Hyperparameter	CIFAR10	SST-2	Speech
θ_0 Pretrained?	✓	✓	
Existing Adv. Dataset			✓
Data Augmentation?	✓		
Validation Split	$\frac{1}{6}$	Predefined	$\frac{1}{6}$
Optimizer	SGD	Adam	SGD
$\lvert\mathcal{D}_{\text{adv}}\rvert$	50	50	11–18[8]
Batch Size	256	32	32
# Epochs	30	4	30
# Subepochs	5	3	3
η (Peak)	$1 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	$1 \cdot 10^{-3}$
η Scheduler	One cycle	Poly. Decay	One cycle
Weight Decay	$1 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	$1 \cdot 10^{-3}$
Dropout Rate	N/A	0.1	N/A

B.2.2. Poison Crafting

Both Zhu et al.'s (2019) and Wallace et al.'s (2021) poison crafting algorithms have their own dedicated hyperparameters, which are detailed in Tables 4 and 5 respectively. Note that Table 5's hyperparameters are taken unchanged from the original source code provided by Wallace et al.

[6] Stanford Sentiment Treebank dataset (SST-2) is used for sentiment analysis

[7] Clean only. Dataset also has 300 backdoored samples divided among the 10 attacks (e.g., $0 \to 1$, $1 \to 2$, etc.).

[8] Varies by digit pair. See Table 2.

Table 4. Convex polytope poison crafting (Zhu et al., 2019) hyperparameter settings

Hyperparameter	Value
# Iterations	1,000
Learning Rate	$4 \cdot 10^{-2}$
Weight Decay	0
Max. Perturb. (ϵ)	0.1

Table 5. SST-2 sentiment analysis poison crafting hyperparameter settings. These are identical to Wallace et al.'s (2021) hyperparameter settings.

Hyperparameter	Value
Optimizer	Adam
Total Num. Updates	20,935
# Warmup Updates	1,256
Max. Sentence Len.	512
Max. Batch Size	7
Learning Rate	$1 \cdot 10^{-5}$
LR Scheduler	Polynomial Decay

B.2.3. BASELINES

Koh & Liang's (2017) influence function method uses Pearlmutter's (1994) stochastic Hessian-vector product (HVP) estimation algorithm. Pearlmutter's algorithm requires 5 hyperparameters; we follow Koh & Liang's notation for these parameters below.

Influence function's five hyperparameters are required to ensure estimator quality and to prevent numerical instability/divergence. Table 6 details the influence function hyperparameters used for each of the Section 4's datasets. t and r were selected to make a single pass through the training set in accordance with the procedure specified by Koh & Liang.

As noted by Basu et al. (2021), influence functions can be fragile on deep networks. We tuned β and γ to prevent HVP divergence, which is common with influence functions.

Table 6. Influence function hyperparameter settings

	Poison		Backdoor
Hyperparameter	CIFAR10	SST-2	Speech
Minibatch Size	1	1	1
Damp (β)	$1 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$5 \cdot 10^{-3}$
Scale (γ)	$3 \cdot 10^{7}$	$1 \cdot 10^{6}$	$1 \cdot 10^{4}$
Recursion Depth (t)	2,500	6,740	1,000
Repeats (r)	10	10	10

Peri et al.'s (2020) Deep k-NN labels a training example as poison if its label does not match the plurality of its neighbors. For Deep k-NN to accurately identify poison, it must generally hold that $k > 2|\mathcal{D}_{\text{adv}}|$. Peri et al. propose selecting k using the *normalized k-ratio*, k/N, where N is the size of the largest class in $\mathcal{D}_{\text{tr}}$.

Peri et al.'s ablation study showed that Deep k-NN generally performed best when the normalized k-ratio was in the range $[0.2,2]$. To ensure a strong baseline, our experiments tested Deep k-NN with three normalized k-ratio values, $\{0.2, 1, 2\}$,[9] and we report the top performing k's result.

[9]This corresponds to $k \in \{833, 4167, 8333\}$ for 25,000 CIFAR10 training examples and a $\frac{1}{6}$ validation split ratio.

B.3. Network Architectures

Table 7 details the CIFAR10 neural network architecture. Specially, we used Page's (2020) ResNet9 architecture, which is the state-of-the-art for fast, high-accuracy (>94%) CIFAR10 classification on DAWNBench (Coleman et al., 2017) at the time of writing.

All sentiment analysis evaluation (see Sec. 4.2) used Liu et al.'s (2020) RoBERTa$_{\text{BASE}}$ pretrained parameters. All language model training used Facebook AI Research's `fairseq` sequence-to-sequence toolkit (Ott et al., 2019) in-line with (Wallace et al., 2021). Text was encoded using Radford et al.'s (2019) *byte-pair encoding* (BPE) scheme.

The speech classification convolutional neural network is identical to that used by Liu et al. (2018) except for two minor changes. First, batch normalization (Szegedy et al., 2013) was used instead of dropout to expedite training convergence. In addition, each convolutional layer's kernel count was halved to allow the model to be trained on a single NVIDIA Tesla K80 GPU.

Table 7. CIFAR10 poison ResNet9 neural network architecture

	Conv1	In=3	Out=64	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=64			
	ReLU				
	Conv2	In=64	Out=128	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=128			
	ReLU				
	MaxPool2D	2 × 2			
↑ ResNet1 ↓	ConvA	In=128	Out=128	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=128			
	ReLU				
	ConvB	In=128	Out=128	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=128			
	ReLU				
	Conv3	In=128	Out=256	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=256			
	ReLU				
	MaxPool2D	2 × 2			
	Conv4	In=256	Out=512	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=512			
	ReLU				
	MaxPool2D	2 × 2			
↑ ResNet2 ↓	ConvA	In=512	Out=512	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=512			
	ReLU				
	ConvB	In=512	Out=512	Kernel=3 × 3	Pad=1
	BatchNorm2D	Out=512			
	ReLU				
	MaxPool2D	2 × 2			
	Linear	Out=10			

Table 8. Speech recognition convolutional neural network

Conv1	In=3	Out=48	Kernel=11×11	Pad=1
MaxPool2D	3×3			
BatchNorm2D	Out=48			
Conv2	In=48	Out=128	Kernel=5×5	Pad=2
MaxPool2D	3×3			
BatchNorm2D	Out=128			
Conv3	In=128	Out=192	Kernel=3×3	Pad=1
ReLU				
BatchNorm2D	Out=192			
Conv4	In=192	Out=192	Kernel=3×3	Pad=1
ReLU				
BatchNorm2D	Out=192			
Conv5	In=192	Out=128	Kernel=3×3	Pad=1
ReLU				
MaxPool2D	3×3			
BatchNorm2D	Out=128			
Linear	Out=10			

C. Additional Evaluation Results

C.1. CIFAR10 Convex Polytope Poisoning Full Results

Poison Identification & Detection Tables 9 and 10 contain the numerical values (including standard deviation) corresponding to the results shown visually in Figures 2 and 4 respectively. Table 11 shows CosIn's aggregated defense success rate.

Table 9. Vision Poison Identification: Poison identification AUPRC mean and standard deviation across >50 trials for four CIFAR10 class pairs with $|\mathcal{D}_{\text{adv}}| = 50$. Both CosIn using just initial parameters θ_0 and with 5 subepoch checkpointing outperformed all baselines for all class pairs. Mean results are shown graphically in Figure 2.

Classes		Ours		Baselines			
Targ.	Pois	CosIn θ_0 Only	CosIn	Deep k-NN	TracIn	Influence Func.	Representer Pt.
Bird	Dog	0.749 ± 0.212	0.859 ± 0.149	0.055 ± 0.155	0.172 ± 0.103	0.116 ± 0.156	0.027 ± 0.013
Dog	Bird	0.850 ± 0.127	0.863 ± 0.117	0.057 ± 0.106	0.197 ± 0.108	0.058 ± 0.056	0.017 ± 0.006
Frog	Deer	0.907 ± 0.125	0.961 ± 0.074	0.184 ± 0.288	0.250 ± 0.136	0.122 ± 0.137	0.024 ± 0.012
Deer	Frog	0.819 ± 0.147	0.881 ± 0.095	0.022 ± 0.054	0.195 ± 0.110	0.081 ± 0.090	0.027 ± 0.020

Table 10. Vision Poison Target Detection: CosIn over only θ_0's top-K z_{targ} detection accuracy across 10 trials for each y_{targ} with 350 clean test examples in each trial. The cutoff threshold was $5Q$ above the median CosIn influence. All values are percentages. Mean results are shown graphically in Figure 4.

Classes		Ours		
Targ.	Pois	Top-1	Top-1%	Top-2%
Bird	Dog	90%	100%	100%
Dog	Bird	100%	100%	100%
Deer	Frog	20%	60%	100%
Frog	Deer	90%	100%	100%

Table 11. Vision Poison Removal: Defense success rate for CosIn (using only θ_0) over 10 trials for three poison removal thresholds. The mean and standard deviation removal percentages over $\mathcal{D}_{\text{adv}}$ and $\mathcal{D}_{\text{cl}}$ demonstrate that our defense removes little clean data.

	$2Q$			$2.5Q$			$3Q$		
	Defense Success	% Poison Removed	% Clean Removed	Defense Success	% Poison Removed	% Clean Removed	Defense Success	% Poison Removed	% Clean Removed
Bird → Dog	100%	95.4 ± 6.5	0.5 ± 0.2	100%	89.0 ± 8.5	0.18 ± 0.10	100%	80.0 ± 12.0	0.06 ± 0.06
Dog → Bird	90%	90.6 ± 10.2	0.2 ± 0.1	90%	80.6 ± 16.4	0.05 ± 0.04	90%	62.2 ± 20.7	0.01 ± 0.02
Deer → Frog	90%	93.8 ± 8.4	0.3 ± 0.1	90%	87.4 ± 12.5	0.05 ± 0.03	90%	77.4 ± 16.4	0.01 ± 0.01
Frog → Deer	100%	86.6 ± 14.8	0.2 ± 0.1	100%	73.0 ± 21.1	0.04 ± 0.03	100%	56.6 ± 24.9	0.01 ± 0.01

C.2. Defending Against a Backdoor Attack on Speech

Liu et al.'s (2018) speech backdoor classification dataset contains spectrograms of human speech pronouncing digits 0 to 9, and includes 300 backdoor training and 200 backdoor test examples.

Liu et al. injected specific background noise as the adversarial trigger. When present, this noise causes a targeted digit to be misclassified as the digit one larger (modulo 10). For instance, a targeted 0 would be misclassified as 1, 1 as 2, etc. making 10 possible digit pairs. Following Liu et al., the underlying model was a speech recognition CNN trained from scratch.

Backdoor Identification Figure 5 compares CosIn (with 3 subepoch checkpointing) to the four baselines across the 10 digit pairs. In all cases, CosIn's mean AUPRC error (i.e., $1 - \text{AUPRC}$) was several times lower than the baselines – in 8/10 cases by more than an order of magnitude. CosIn even achieve perfect identification (i.e., AUPRC $= 1$) for three digit pairs: $0 \to 1$, $3 \to 4$, and $4 \to 5$.

Backdoor Target Detection Table 12 details CosIn's ability to detect whether a particular test example is an attack target. Specifically, an example was classified as "targeted" if any training example's CosIn influence exceeded the median by cQ ($c \in \{4{,}5\}$), with c tunable to control sensitivity. With $c = 4$, there were no false negatives for any digit pair (i.e., precision $= 1$), but the false positive rate (FPR) was larger. Six digit pairs had perfect target detection precision even at $c = 5$.

As expected, those digit pairs where identification performed worse (e.g., $2 \to 3$ and $6 \to 7$) also had poorer target detection. Observe that digit pairs $5 \to 6$ and $9 \to 0$ had near perfect identification but had a higher false positive rate at $c = 4$ and a higher false negative rate at $c = 5$ respectively; such divergent behavior indicates that detection performance could be improved via a better underlying assumptions for Q, e.g., accounting for the non-normality visible in Figures 1d to 1f. We leave this to future work.

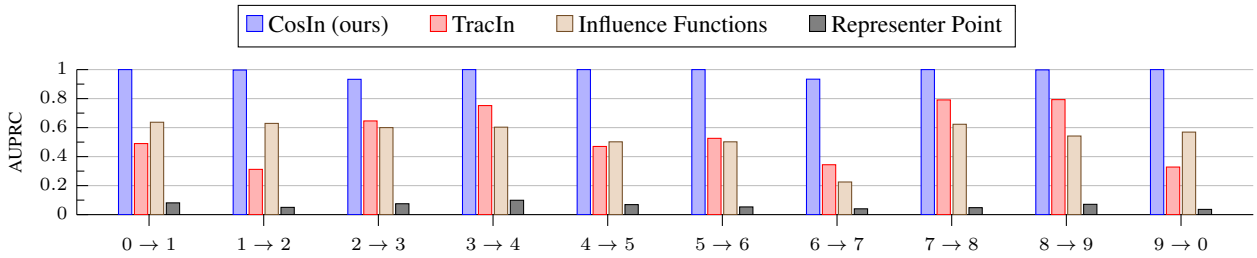


Figure 5. Speech Backdoor Identification: Mean speech backdoor example identification AUPRC across 30 trials for $11 \leq |\mathcal{D}_{\text{adv}}| \leq 18$ (varies by class pair). CosIn with 3 subepoch checkpointing always outperformed the baselines – often by an order of magnitude. For digit pairs $0 \to 1$, $3 \to 4$, and $4 \to 5$, CosIn always perfectly identified $\mathcal{D}_{\text{adv}}$ (i.e., AUPRC $= 1$). See Table 13 for numerical results including standard deviation.

Table 12. Speech Backdoor Target Detection: CosIn's (with 10 subepoch checkpointing) target, z_{targ}, detection performance across >100 trials for each digit pair at cutoff thresholds of $4Q$ and $5Q$ above the median CosIn influence. All values are percentages.

Cutoff	Metric	$0 \to 1$	$1 \to 2$	$2 \to 3$	$3 \to 4$	$4 \to 5$	$5 \to 6$	$6 \to 7$	$7 \to 8$	$8 \to 9$	$9 \to 0$
$4Q$	Acc.	99.2	97.3	100	99.2	97.6	88.5	95.1	95.7	97.4	100
	Prec.	100	100	100	100	100	100	100	100	100	100
	FPR	0.8	2.9	0	0.9	2.8	13.4	5.4	4.9	2.9	0
$5Q$	Acc.	100	99.1	99.4	99.2	100	97.3	97.6	97.9	99.1	96.9
	Prec.	100	100	95.2	100	100	100	83.3	100	91.7	76.5
	FPR	0	1.0	0	0.9	0	3.2	0.9	2.5	0	0

Table 13. Speech Backdoor Identification: Mean and standard deviation AUPRC across 30 trials for Liu et al.'s (2018) speech backdoor dataset with $11 \leq |\mathcal{D}_{\text{adv}}| \leq 18$. COSIN with 3 subepoch checkpointing always outperformed the baselines, even achieving perfect backdoor identification for three digit pairs.

Digits		Ours	Baselines		
Targ	Pois	COSIN	TracIn	Influence Func.	Representer Pt.
0	1	$\mathbf{1 \pm 0}$	1.490 ± 0.217	0.637 ± 0.270	0.081 ± 0.063
1	2	$\mathbf{0.997 \pm 0.007}$	0.312 ± 0.127	0.629 ± 0.174	0.050 ± 0.017
2	3	$\mathbf{0.933 \pm 0.053}$	0.646 ± 0.218	0.600 ± 0.231	0.075 ± 0.057
3	4	$\mathbf{1 \pm 0}$	0.752 ± 0.193	0.603 ± 0.261	0.099 ± 0.104
4	5	$\mathbf{1 \pm 0}$	0.470 ± 0.169	0.502 ± 0.221	0.069 ± 0.070
5	6	$\mathbf{1.000 \pm 0.001}$	0.526 ± 0.133	0.502 ± 0.177	0.053 ± 0.031
6	7	$\mathbf{0.934 \pm 0.074}$	0.344 ± 0.145	0.225 ± 0.150	0.040 ± 0.009
7	8	$\mathbf{1.000 \pm 0.002}$	0.791 ± 0.148	0.623 ± 0.191	0.048 ± 0.021
8	9	$\mathbf{0.998 \pm 0.013}$	0.793 ± 0.204	0.542 ± 0.284	0.071 ± 0.075
9	0	$\mathbf{1.000 \pm 0.001}$	0.328 ± 0.143	0.569 ± 0.276	0.036 ± 0.016

C.3. Sentiment Analysis Poisoning Full Results

Table 14. Natural Language Poison Identification: Poison identification AUPRC mean and standard deviation across 10 trials for 4 positive and 4 negative sentiment SST-2 movie reviews (Socher et al., 2013) with $|\mathcal{D}_{\text{adv}}| = 50$. COSIN with 3 subepoch checkpointing perfectly identified all poison in all but two trials. Mean results are shown graphically in Figure 3.

Review		Ours		Baselines		
Type	No.	COSIN θ_0 Only	COSIN	TracIn	Influence Func.	Representer Pt.
Pos. ↑	1	0.762 ± 0.111	$\mathbf{1 \pm 0}$	0.089 ± 0.037	0.023 ± 0.027	0.003 ± 0.001
	2	0.477 ± 0.098	$\mathbf{1 \pm 0}$	0.151 ± 0.079	0.005 ± 0.002	0.002 ± 0.000
	3	0.852 ± 0.062	$\mathbf{1 \pm 0}$	0.039 ± 0.039	0.002 ± 0.001	0.001 ± 0.001
Pos. ↓	4	0.435 ± 0.054	$\mathbf{1 \pm 0}$	0.011 ± 0.003	0.003 ± 0.004	0.001 ± 0.000
Neg. ↑	1	0.518 ± 0.082	$\mathbf{1 \pm 0}$	0.005 ± 0.001	0.005 ± 0.003	0.002 ± 0.001
	2	0.920 ± 0.081	$\mathbf{1 \pm 0}$	0.279 ± 0.060	0.006 ± 0.004	0.001 ± 0.001
	3	0.255 ± 0.075	$\mathbf{0.983 \pm 0.055}$	0.065 ± 0.035	0.003 ± 0.002	0.001 ± 0.001
Neg. ↓	4	0.304 ± 0.087	$\mathbf{1 \pm 0}$	0.007 ± 0.001	0.003 ± 0.001	0.001 ± 0.001

C.4. Adversarial Set Identification Execution Time

Table 15 compares the execution time of COSIN to the other general-purpose, influence-based benchmarks across 50 trials. All results were collected on an HPC system with 3 Intel E5-2690v4 cores, 48GB of 2400MHz DDR4 RAM, and a single NVIDIA Tesla K80.

With the exception of the representer point baseline (Yeh et al., 2018) which considers only the network's linear classification layer (and had the worst performance in our experiments), COSIN using only the initial θ_0 parameters was the fastest – by a large margin. Note that COSIN (see Alg. 1) is also embarrassingly parallel and can be naively sped up across multiple GPUs/clusters.

Recall that Deep k-NN Peri et al. (2020) is an empirical, defense against clean-label data poisoning and serves as an additional baseline for Section 4.1's convex polytope attack experiments on CIFAR10. Across >50 trials using the above hardware setup, Deep k-NN's execution time had mean and standard deviation of 242 seconds and 1.1 seconds respectively.

Table 15. Adversarial Set Identification Execution Time: Mean and standard deviation algorithm execution time (in seconds) across >50 trials for: convex polytope poisoning attack on CIFAR10, sentiment analysis poisoning attack on SST-2, and speech classification using Liu et al.'s (2018) backdoored dataset. COSIN (using only θ_0) is multiple times faster than TracIn and influence functions. Representer point only directly consider the network's final linear layer so its execution time cannot be directly compared to the other methods.

Type	Dataset	COSIN θ_0 Only	COSIN	TracIn	Influence Func.	Representer Pt.
Image	CIFAR10	212 ± 2	$24{,}267 \pm 1{,}939$	$5{,}910 \pm 600$	$15{,}634 \pm 641$	187 ± 2
Sentiment	SST-2	$4{,}213 \pm 17$	$27{,}723 \pm 7{,}933$	$16{,}667 \pm 187$	$21{,}409 \pm 77$	$1{,}697 \pm 11$
Speech	(Liu et al., 2018)	N/A	$2{,}605 \pm 710$	894 ± 279	$4{,}595 \pm 177$	49 ± 1