[Supplementary Material] Mitigating Backdoor Attacks using Activation-Guided Model Editing

Felix Hsieh[1,2], Huy H. Nguyen[1], AprilPyone MaungMaung[1], Dmitrii Usynin[2,3], and Isao Echizen[1,4]

[1] National Institute of Informatics, Tokyo, Japan
[2] Technical University of Munich, Munich, Germany
[3] Imperial College London, London, United Kingdom
[4] The University of Tokyo, Tokyo, Japan

A Experimental Setup

A.1 System Hardware

We used three different setups for our experiments due to changing resource accessibility. First one with 251 GiB system memory, four Intel Xeon E5-2698 v4 CPUs [3], and one NVIDIA Tesla V100 [8]. The second one with 264 GiB system memory, 36 Intel Core i9-10980XE CPUs [2], and one NVIDIA RTX A6000 [7]. The third one contained 16 GiB system memory, one AMD Ryzen 5 5600X CPU [1], and one Nvidia RTX 4600 Ti GPU [6].

B Additional Results

B.1 Sample and Time Efficency

This section expands upon the limited results shown in Table 6. Table 7 shows the full results of the experiment and, specifically, the performance of our method when we use repairing. With Backdoor Knowledge (BDK), repairing required 5000 samples when aiming to increase Clean Target Class Accuracy (CTCA) while maintaining a relatively low Attack Success Rate (ASR). With ¬BDK, repairing had a positive effect on CTCA with 500 samples, but with 5000, the trade-off between unlearning and target class utility is better.

The baseline methods depended heavily on a certain number of samples to achieve method-specific top performance. Certain baseline methods, such as actFT and basicFT, seem to require a high number of samples to have any effect in reducing ASR or maintaining ACC above random guessing. Without repairing, our method performed similarly for different test runs with different-sized unlearning data.

Notably, when comparing the results from actFT and basicFT with five samples, we can see a problem with our current scoring function. While actFT, with a score close to 0, has not unlearned the backdoor but maintains model utility,

Table 7: Efficiency of proposed unlearning compared with state-of-the-art methods. We experimented with 50%(5000), 5%(500), 0.5%(50), and 0.05%(5) of unseen CIFAR10 data for unlearning. Best results are highlighted in bold.

Method	Number of Samples	Metric				
		Score (↑)	ASR (↓)	ACC (↑)	CTCA (↑)	Time (↓)
actFT [9]	5000	**92.95±4.72**	5.28±2.48	**69.64±0.92**	58.1±9.1	3.96±0.34
	500	7.0±2.09	87.17±2.86	**69.41±1.02**	**61.47±7.99**	2.81±0.04
	50	0.21±0.06	93.67±1.59	**69.81±1.04**	**61.42±8.41**	3.01±0.5
	5	0.22±0.08	93.66±1.6	**69.82±1.06**	**61.58±8.38**	2.44±0.03
BaEraser [5]	5000	57.47±15.65	2.0±2.79	41.81±12.3	14.79±20.77	623.3±111.46
	500	80.69±1.93	3.81±2.41	59.45±0.83	32.98±1.1	138.88±2.87
	50	54.48±13.21	24.16±16.46	51.83±2.66	39.64±8.6	21.35±0.17
	5	16.88±11.53	43.01±34.71	19.06±4.29	56.76±31.71	20.23±0.2
Ours(BDK)	5000	**92.38±1.67**	**0.0±0.0**	65.39±0.42	0.0±0.0	**0.81±0.05**
	500	**92.3±1.45**	**0.0±0.0**	65.2±0.58	0.0±0.0	**0.58±0.01**
	50	**92.4±1.39**	**0.0±0.0**	65.29±0.58	0.0±0.0	**0.38±0.01**
	5	**92.51±1.34**	**0.0±0.0**	65.37±0.56	0.0±0.0	**0.38±0.02**
Ours(BDK) +repair	5000	77.95±1.6	9.1±1.08	61.11±0.89	43.98±3.66	49.66±0.53
	500	76.63±3.63	**0.01±0.0**	54.14±2.4	0.0±0.0	67.7±1.76
	50	83.64±2.81	**0.0±0.0**	59.09±1.82	0.0±0.0	72.02±0.98
	5	67.56±8.98	**0.0±0.0**	47.69±5.95	0.0±0.0	70.39±1.25
basicFT	5000	69.49±1.11	6.0±1.07	52.57±0.91	33.86±4.07	71.56±1.47
	500	14.14±0.1	**0.0±0.0**	9.99±0.03	0.0±0.0	73.34±0.2
	50	14.13±0.1	**0.0±0.0**	9.98±0.02	0.0±0.0	71.59±0.25
	5	14.17±0.11	**0.0±0.0**	10.01±0.0	0.0±0.0	71.49±0.03
NAD [4]	5000	71.23±3.2	4.54±1.48	52.95±1.41	32.05±1.3	114.9±1.45
	500	42.18±5.42	6.68±2.06	32.09±3.97	23.18±9.4	117.69±1.9
	50	42.49±7.39	9.36±5.28	33.58±6.54	20.92±11.71	111.91±1.06
	5	7.44±8.48	59.03±41.63	11.71±1.64	**63.93±37.39**	118.7±5.99
Ours(¬BDK)	5000	82.67±0.51	**0.0±0.0**	58.53±0.97	0.0±0.0	**0.81±0.02**
	500	82.8±0.53	**0.0±0.0**	58.49±0.21	0.0±0.0	**0.51±0.06**
	50	**89.9±2.02**	**0.0±0.0**	**63.52±1.24**	0.0±0.0	**0.36±0.02**
	5	**89.5±1.72**	**0.0±0.0**	**63.23±0.86**	0.0±0.0	**0.4±0.01**
Ours(¬BDK) +repair	5000	77.47±2.76	10.79±4.74	62.09±0.81	51.07±4.78	49.4±0.17
	500	56.7±14.54	24.66±18.58	54.72±2.02	45.26±9.69	68.96±0.33
	50	83.53±1.0	**0.0±0.0**	59.02±0.64	0.0±0.0	73.86±1.75
	5	62.18±4.95	**0.04±0.04**	43.93±3.13	0.07±0.09	74.43±0.43
Retraining	47500	-	4.03±0.99	70.27± 0.66	60.55±4.3	423.56±73.96

basicFT achieves a higher score with a model that has completely lost its utility. Our current scoring function prioritizes successful backdoor mitigation over maintaining model utility. Additionally, the scoring function does not emphasize tracking the complete loss of utility for a single class. Maintaining at least a decent ACC for each class is crucial to ensuring that a model retains its essential utility.

B.2 Mitigating Limitation with Repairing

This experiment shows how the learning rate influences repairing with one epoch. We aim for a suitable learning rate to restore utility without sacrificing too much backdoor unlearning capability. We are limited to finetune only on unseen unlearn data. This limitation can lead to reduced generalizability compared to the original model because D_U is one order of magnitude smaller than D_T.

Table 8: Repairing performance with different values for learning rate η. For comparison, we include the model state before unlearning (Before) and after weight editing without repairing (None). Best results are highlighted in bold.

	η	ASR (↓)	ACC (↑)	CTCA (↑)
Before	-	94.53±1.29	70.5±0.18	61.39±8.51
BDK	None	**0.0±0.0**	65.49±0.39	0.0±0.0
	1e-6	**0.0±0.0**	62.29±0.09	0.0±0.0
	1e-5	**0.0±0.0**	62.6±0.1	0.0±0.0
	1e-4	**0.0±0.0**	64.08±0.31	0.0±0.0
	1e-3	17.83±5.28	**67.56±0.23**	**47.23±2.78**
	1e-2	**9.79±0.88**	59.87±3.04	**42.07±7.72**
	1e-1	**0.3±0.43**	20.44±1.56	0.87±1.23
¬BDK	None	**0.3±0.21**	59.18±2.63	0.14±0.19
	1e-6	28.77±23.08	**62.59±0.41**	3.3±3.85
	1e-5	32.03±24.75	**63.18±0.56**	5.77±5.66
	1e-4	43.17±30.91	**66.08±1.47**	20.56±14.76
	1e-3	53.48±12.29	**68.47±0.17**	**55.06±2.67**
	1e-2	**23.09±8.07**	**61.83±1.33**	**55.02±4.41**
	1e-1	**6.01±3.54**	24.57±0.38	13.66±8.73

Table 8 shows repairing can help restore CTCA while risking increasing the ASR and potentially decreasing the ACC for other classes. The effectiveness of repairing depends heavily on the used dataset and poisoned trigger. In this experiment, the learning rate 1e-2 was the most effective when aiming for a decent CTCA and a low ASR, achieving a good balance between forgetting and utility.

B.3 Effect of Activation Mean and Standard Deviation

This experiment shows the best values for mean and standard deviation hyperparameters λ and γ for achieving effective unlearning. We also show how changing the Batch Normalization (BN) Moving Average (MA) parameters during activation extraction affects the unlearning performance.

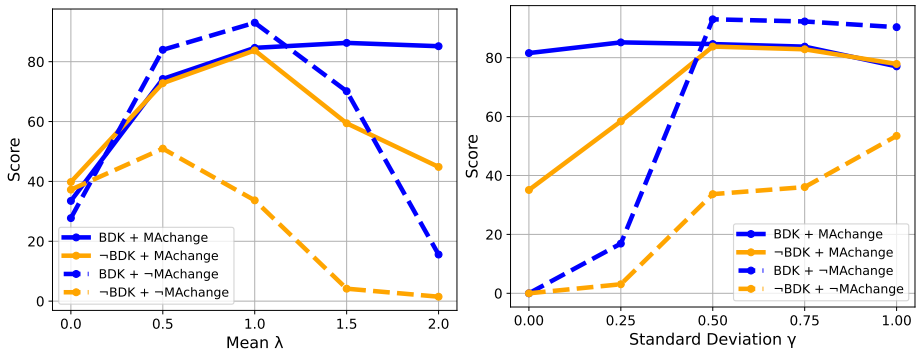

Fig. 4: Performance of our method when using different values for hyperparameters λ and γ. Compared with or without change in BN MA parameters.

Figure 4 shows that allowing change of MAs can improve performance when we unlearn with ¬BDK. For unlearning with BDK, we can achieve the best performance when we keep the MAs fixed. When using the respective better-performing process, we get the best performance with 1.0 as a value for λ.

Parameter γ mainly influences the impact of the unlearning process on the model weights. A higher value for γ generally means a more invasive model editing and a substantial weight change of those neurons that reacted strongly to the input dataset.

With BDK, we achieve the best performance with fixed MAs. Without BDK, allowing change of MAs is better. Rescaling with a mean of 1.0 and standard deviation of 0.5 is most suitable for stable, high results. When γ equals 0, we can observe the isolated influence of the change in MA parameters. The reason is that the influence of model editing is nonexistent when activations are scaled to 0 before shifting every activation value to 1.

B.4 Effects of different Activations

In this experiment, we show how the unlearning performs when we choose a different activation formula for Equation 1. Instead of using just the negative activation of the unlearn dataset D_U, we introduce two hyperparameters α and β that balance out the activation with clean and poisoned D_U. In the case of

¬BDK, where we have no trigger for poisoning, clean and poisoned activation are the same. The formula that replaces Equation 1 is calculated as

$$A = \alpha \cdot A_{\text{clean}} + \beta \cdot A_{\text{poisoned}}. \tag{7}$$

Table 9: Score (↑) comparison for different values for hyperparameter α and β. Best results are highlighted in bold.

	β	α		
		-1	0	1
BDK	-1	**93.38±0.58**	**93.02±0.66**	**92.89±1.17**
	0	33.67±42.76	14.34±0.16	0.0±0.0
	1	0.0±0.0	0.0±0.0	0.0±0.0
¬BDK	-1	**86.67±1.43**	**86.67±1.43**	14.34±0.16
	0	**86.67±1.43**	14.34±0.16	0.62±0.88
	1	14.34±0.16	0.62±0.88	0.62±0.88

Table 9 shows that our method performs well in both unlearning scenarios when we set β to -1 and α to either -1 or 0. For the final algorithm, we decided to use values 0 and -1 for α and β, respectively, because setting at least one hyperparameter to 0 reduces the complexity of the function and allows us to extract one activation set less.

B.5 Target Class Dependencies

In this experiment, we compare the performance of our methods with the baseline methods when we use different backdoor target classes. The robustness of the classes can vary significantly. We want to show that our method achieves consistency for all classes in CIFAR10.

Table 10 shows that our method performs better on average when we have ¬BDK. With BDK, the actFT proposed by [9] performs slightly better on average.

References

1. AMD: Amd ryzen™ 5 5600x desktop processors, `https://www.amd.com/en/products/processors/desktops/ryzen/5000-series/amd-ryzen-5-5600x.html`, accessed on July 5th, 2024

Table 10: Score (↑) comparison of different methods based on the backdoor target class. Best results are highlighted in bold.

Target	BDK			¬BDK		
Class	actFT [9]	BaEraser [5]	Ours	basicFT	NAD [4]	Ours
0	**94.85±0.57**	49.24±10.22	90.1±0.64	70.66±5.0	68.71±3.77	**80.82±2.44**
1	**94.49±1.49**	43.47±21.12	88.28±0.35	58.32±6.92	68.26±0.72	**75.57±0.62**
2	**94.62±2.61**	59.5±4.62	93.02±0.66	61.9±3.63	65.33±3.04	**83.77±3.34**
3	**91.21±1.83**	31.77±10.03	94.41±0.17	49.96±11.41	65.98±5.62	**87.46±0.11**
4	**93.71±3.22**	57.17±25.75	90.61±2.5	62.76±5.06	68.32±1.17	**87.63±0.5**
5	**96.15±3.24**	61.39±15.54	93.78±1.7	57.12±18.84	69.7±4.58	**89.67±0.77**
6	**93.05±2.45**	38.2±20.33	89.08±0.83	55.08±18.59	67.38±3.96	**81.53±2.21**
7	**95.86±2.12**	29.57±14.03	90.78±1.76	70.93±3.53	71.59±4.47	**77.7±3.05**
8	**93.37±2.73**	35.28±5.39	87.23±1.2	30.56±23.24	66.2±2.52	**77.32±0.47**
9	**92.08±3.09**	53.83±13.81	90.57±0.91	58.7±13.01	70.75±5.21	**85.46±1.6**
Average	**93.94±2.33**	45.94±14.08	90.79±1.07	57.6±10.92	68.22±3.51	**82.69±1.51**

2. Intel: Intel® core™ i9-10980xe extreme edition processor, https://www.intel.com/content/www/us/en/products/sku/198017/intel-core-i910980xe-extreme-edition-processor-24-75m-cache-3-00-ghz/specifications.html, accessed on April 16th, 2024
3. Intel: Intel® xeon® processor e5-2698 v4, https://www.intel.com/content/www/us/en/products/sku/91753/intel-xeon-processor-e52698-v4-50m-cache-2-20-ghz/specifications.html, accessed on Feburary 27th, 2024
4. Li, Y., Lyu, X., Koren, N., Lyu, L., Li, B., Ma, X.: Neural attention distillation: Erasing backdoor triggers from deep neural networks (2021)
5. Liu, Y., Fan, M., Chen, C., Liu, X., Ma, Z., Wang, L., Ma, J.: Backdoor defense with machine unlearning (2022)
6. NVIDIA: Geforce rtx 4060 family, https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4060-4060ti/, accessed on July 5th, 2024
7. NVIDIA: Nvidia rtx a6000 graphics card, https://www.nvidia.com/en-us/design-visualization/rtx-a6000/, accessed on April 16th, 2024
8. NVIDIA: Nvidia tesla v100 gpu accelerator, https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, accessed on Feburary 27th, 2024
9. Qiao, X., Yang, Y., Li, H.: Defending neural backdoors via generative distribution modeling (2019)