Safe Robot Learning in Assistive Devices through Neural Network Repair - Supplementary Materials

Keyvan Majd[1†], **Geoffrey Clark**[1], **Tanmay Khandait**[1], **Siyu Zhou**[1],
Sriram Sankaranarayanan[2], **Georgios Fainekos**[3], **Heni Ben Amor**[1]

[1]Arizona State University, [2]University of Colorado Boulder, [3] Toyota NA-R&D

[†]`majd@asu.edu`

A Interval Arithmetic Method

To illustrate how we generated a tight valid bound for each ReLU activation node, we used the Interval Arithmetic method [1, 2]. Interval arithmetic is widely used in verification to find an upper and a lower bounds over the relaxed ReLU activations given a bounded set of inputs. We used the same approach to find the tight bounds over the ReLU nodes assuming the weights can only perturb inside a bounded l_∞ error with respect to the original weights. Assume we denote each input variable of repair layer L as $x^{L-1}(i)$, the weight term that connect $x^{L-1}(i)$ to $x^L(j)$ as $\theta_w^L(ij)$, and the bias term of nodes $x^L(j)$ as $\theta_b^L(j)$. Given the bounds for the variables $\theta_w^L(ij) \in \left[\underline{\theta}_w^L(ij), \bar{\theta}_w^L(ij)\right]$ and $\theta_b^L(ij) \in \left[\underline{\theta}_b^L(ij), \bar{\theta}_b^L(ij)\right]$, the interval arithmetic gives the valid upper and lower bounds for $x^L(j)$ as

$$\bar{x}^L(j) = \sum_i \left(\bar{\theta}_w^L(ij)\max(0, x^{L-1}(i)) + \underline{\theta}_w^L(ij)\min(0, x^{L-1}(i))\right) + \bar{\theta}_b^L(ij), \text{ and}$$
$$\underline{x}^L(j) = \sum_i \left(\underline{\theta}_w^L(ij)\max(0, x^{L-1}(i)) + \bar{\theta}_w^L(ij)\min(0, x^{L-1}(i))\right) + \underline{\theta}_b^L(ij),$$

respectively. The bounds over the ReLU nodes in the subsequent layers $l = L+1, \cdots N$ are obtained as

$$\bar{x}^l(j) = \sum_i \left(\bar{x}^{l-1}\max(0, \theta_w^l(ij)) + \underline{x}^{l-1}\min(0, \theta_w^l(ij))\right) + \theta_b^l(ij),$$
$$\underline{x}^l(j) = \sum_i \left(\underline{x}^{l-1}\max(0, \theta_w^l(ij)) + \bar{x}^{l-1}\min(0, \theta_w^l(ij))\right) + \theta_b^l(ij).$$

B Soundness of NNRepLayer

Our technique only guarantees the satisfaction of constraints for the repaired samples. While we empirically showed the generalizability of our technique, our method does not guarantee the satisfaction of constraints for the unseen adversarial samples. To address this problem, we propose Algorithm 1 that guarantees the satisfaction of constraints Ψ for the inputs $x \in \mathcal{X}_r$. In this algorithm, NNReplayer is employed with a sound verifier [3, 4] in the loop such that our method first returns the repaired network π_θ^r. Then, the verifier evaluates the network. If the algorithm terminates, the network is guaranteed to be safe for all other unseen samples in the target input space $\mathcal{X}_r$. Otherwise, the network is not satisfied to be safe and the verifier provides the newly found adversarial samples $\mathcal{X}_r$ for which the guarantees do no hold. In turn, NNRepLayer uses the given samples by the verifier to repair the network. This loop terminates when the verifier confirms the satisfaction of constraints.

Algorithm 1 NNRepLayer in a loop with a sound verifier

```
Input: $\pi_\theta^o, \mathcal{X}_r, \Psi$
Output: $\pi_\theta^r$
1: $\pi_\theta^r \leftarrow \pi_\theta^o$
2: while $\mathcal{X}_r \notin \emptyset$ do
3:     $\pi_\theta^r \leftarrow \text{NNRepLayer}(\pi_\theta^r, \mathcal{X}_r, \Psi)$
4:     $\mathcal{X}_r \leftarrow \text{Verifier}(\pi_\theta^r)$
5: end while
```

Theorem 1. *Assume* VERIFIER() *is a sound verifier [3, 4]. If the Algorithm 1 terminates, the predicate Ψ is satisfied by the repaired network π_θ^r.*

Proof. Given the sound verifier VERIFIER(), if the algorithm terminates, $\mathcal{X}_r$ is empty which means VERIFIER() did not find other samples that violate Ψ. Therefore, the predicate Ψ is guaranteed to be satisfied by π_θ^r. $\square$

C Comparing NNRepLayer with the Other Repair Techniques

We run the repair codes from REASSURE [5] and PRDNN [6] on our problem. To ensure the accuracy of formulated repair using the technique presented in [5], we contacted the authors. Running our control tasks on the code provided in [6] returns error. We realized the error is due to the applicability of [6] only for the networks with one and two dimensional inputs (as the authors also mentioned in the paper [6]). As shown in Fig. 4 (of the paper), REASSURE [5] accurately satisfies the bounding constraint. However, it introduces controls with large magnitudes in the repaired regions. Fig. 6 (of the paper) shows that REASSURE [5] fails to satisfy the input-output constraints while our method satisfies these constraints. Overall, the repair methods [5, 6] are either infeasible to be applied to the network sizes we used, or perform poorly and cannot accommodate the complicated constraints that we address.

Table 1 shows the detailed comparison between our technique and the other repair methods in the literature. Table 1 and our empirical evaluations illustrate that our method is the only repair method in the literature that can impose complicated hard constraints to the network.

Table 1: Comparing our technique with the other methods in the literature

	Train SAT Guarantee	Generalization	Side Effect	Complicated Hard Constraints	Hidden Layer Repair
Fine-tune and Retrain [7, 8, 9]	No	No	Yes	No	Yes
Arch. Extension [5, 6]	Yes	No	Yes	No	— (functional space)
Goldberg et al. [10]	Yes	No	Yes	No	No
Our method	Yes	No	Yes	Yes	Yes

D Testing Repair on Larger Networks

To demonstrate the scalability of our method, we conducted a repair experiment on a network with 256 neurons in each hidden layer. We used 1000 samples for repair and 2000 samples for testing. We formulate this problem in MIQP and run the program on a Gurobi [11] solver. We terminated the solver after 10 hours and report the best found feasible solution. Figure 1 and Table 2 show the control signal, and the statistical results of this experiment, respectively. As demonstrated, our technique repaired a network with up to 256 nodes with 100% repair efficacy in 10 hours.

Table 2: Experimental results for repairing a network with 256 nodes in each hidden layer for the input-output constraint repair, maximum ankle angle rate of 2 [rad/s]. The table reports the size of network, the number of samples, the Mean Absolute Error (MAE) between the repaired and the original outputs, the percentage of adversarial samples that are repaired (Repair Efficacy), and the runtime.

	Network Size	Number of Samples	MAE	Repair Efficacy [%]	Runtime [h]
Input-output Constraint	256	1000	0.62	100	10

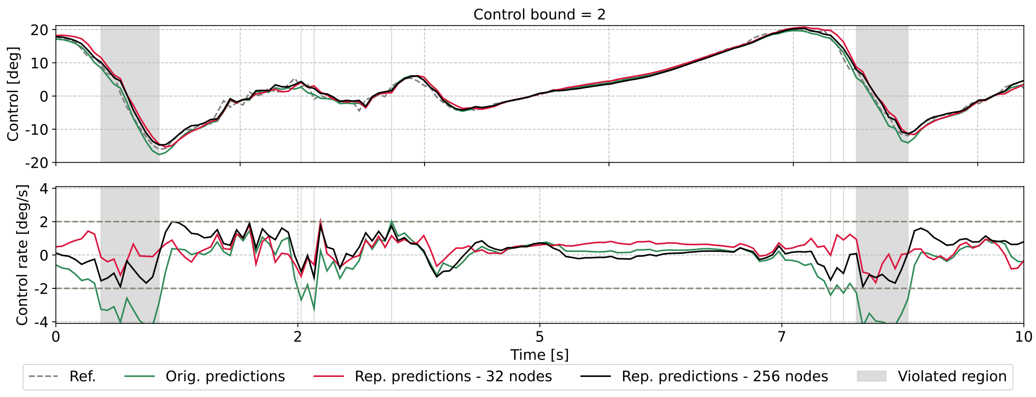

Figure 1: Input-output constraints for networks with 32 (red) and 256 (black) nodes in each hidden layer: Ankle angles and Ankle angle rates for bounds $\Delta\alpha_a = 2$.

E Repairing a Single Layer Partially

In this section, we provide examples for the computational speedups and heuristics that lead to faster neural network repair by repairing randomly selected nodes of a single hidden layer in a network with 64 hidden nodes for 35 times. We let the solver run for 30 minutes in each experiment (versus the full repair that is solved in 6 hours).

Fig. 2 demonstrates the mean absolute error (MAE), the total number of repaired weights, repair efficacy, and the original MIQP cost. To impose sparsity to the weight changes of the repair layer, we solved the original full repair by adding the l_1 norm-bounded error of repaired weights with respect to their original values to the MIQP cost function. The bold bars in Fig. 2 demonstrate the results of repairing the 10 randomly-selected sparse nodes. Repairing of the obtained sparse nodes reached a cost value very close to the cost value of the originally full repair problem (42.99 versus 40.29), **in only 30 minutes** versus 6 hours. As illustrated, repair of some random nodes also results in infeasibility (blank bars) that shows these nodes cannot repair the network. In our future work, we aim to explore the techniques such as neural network pruning [12, 13], to select and repair just the layers and nodes that can satisfy the constraints instead of repairing a full layer. Our results in this experiment show that repairing partial nodes can significantly decrease the computational time of our technique.

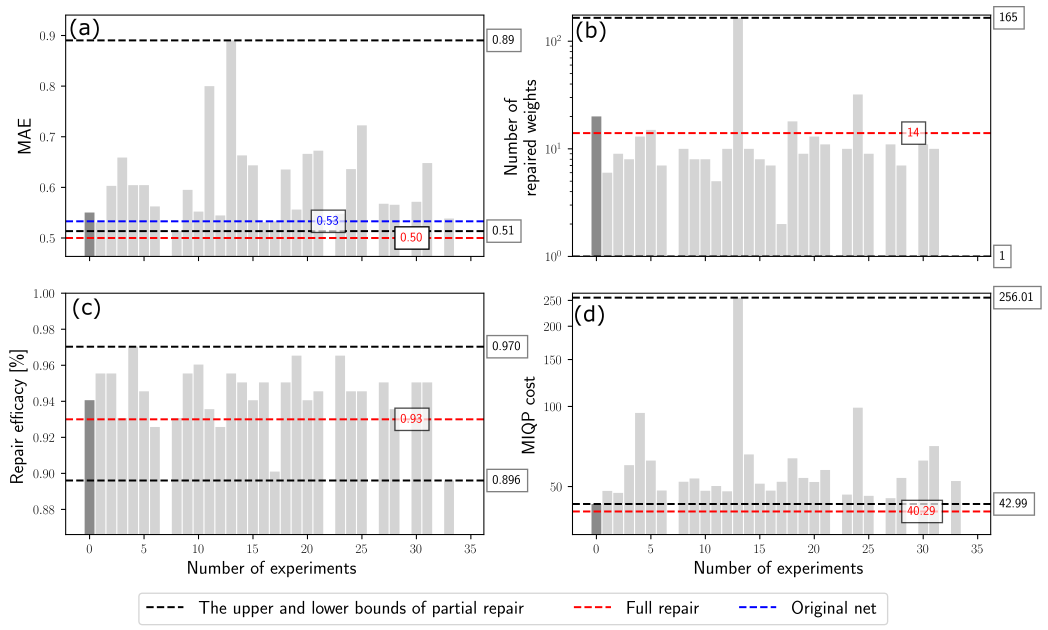

Figure 2: Partial repair versus full repair: we repaired 10 randomly selected nodes for 30 minutes in a network with 64nodes in each hidden layer (a) mean absolute error (MAE), (b) the total number of repaired weights, (c) repair efficacy, and (d) MIQP cost. We performed this random selections for 35 times.

References

[1] R. E. Moore, R. B. Kearfott, and M. J. Cloud. Introduction to interval analysis/ramon e. *Moore, R. Baker Kearfott, Michael J. Cloud. Philadelphia*, 2009.

[2] V. Tjeng, K. Y. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[3] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[4] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[5] F. Fu and W. Li. Sound and complete neural network repair with minimality and locality guarantees. *arXiv preprint arXiv:2110.07682*, 2021.

[6] M. Sotoudeh and A. V. Thakur. Provable repair of deep neural networks. In *42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 588–603, 2021.

[7] A. Sinitsin, V. Plokhotnyuk, D. Pyrkin, S. Popov, and A. Babenko. Editable neural networks. In *International Conference on Learning Representations*, 2019.

[8] X. Ren, B. Yu, H. Qi, F. Juefei-Xu, Z. Li, W. Xue, L. Ma, and J. Zhao. Few-shot guided mix for dnn repairing. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 717–721. IEEE, 2020.

[9] G. Dong, J. Sun, X. Wang, X. Wang, and T. Dai. Towards repairing neural networks correctly. In *IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 714–725, 2021.

[10] B. Goldberger, G. Katz, Y. Adi, and J. Keshet. Minimal modifications of deep neural networks using verification. In *23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73, pages 260–278, 2020.

[11] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL https://www.gurobi.com.

[12] B. Hassibi, D. Stork, and G. Wolff. Optimal brain surgeon: Extensions and performance comparisons. *Advances in neural information processing systems*, 6, 1993.

[13] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.